SPARQL for a Web of Linked Data: Semantics and Computability

(Extended Version)*

Olaf Hartig

Humboldt-Universität zu Berlin
hartig@informatik.hu-berlin.de

Abstract. The World Wide Web currently evolves into a Web of Linked Data where content providers publish and link data as they have done with hypertext for the last 20 years. While the declarative query language SPARQL is the de facto for querying a-priory defined sets of data from the Web, no language exists for querying the Web of Linked Data itself. However, it seems natural to ask whether SPARQL is also suitable for such a purpose.
In this paper we formally investigate the applicability of SPARQL as a query language for Linked Data on the Web. In particular, we study two query models: 1) a *full-Web semantics* where the scope of a query is the complete set of Linked Data on the Web and 2) a family of *reachability-based semantics* which restrict the scope to data that is reachable by traversing certain data links. For both models we discuss properties such as monotonicity and computability as well as the implications of querying a Web that is infinitely large due to data generating servers.

1 Introduction

The emergence of vast amounts of RDF data on the WWW has spawned research on storing and querying large collections of such data efficiently. The prevalent query language in this context is SPARQL [16] which defines queries as functions over an RDF dataset, that is, a fixed, a-priory defined collection of sets of RDF triples. This definition naturally fits the use case of querying a repository of RDF data copied from the Web.

However, most RDF data on the Web is published following the Linked Data principles [5], contributing to the emerging Web of Linked Data [6]. This practice allows for query approaches that access the most recent version of remote data on demand. More importantly, query execution systems may automatically discover new data by traversing data links. As a result, such a system answers queries based on data that is not only up-to-date but may also include initially unknown data. These features are the foundation for true serendipity, which we regard as the most distinguishing advantage of querying the Web itself, instead of a predefined, bounded collection of data.

While several research groups work on systems that evaluate SPARQL basic graph patterns over the Web of Linked Data (cf. [9], [10,12] and [13,14]), we notice a shortage of work on theoretical foundations and properties of such queries. Furthermore, there is a need to support queries that are more expressive than conjunctive (basic graph pattern based) queries [17]. However, it seems natural to assume that SPARQL could be used in this context because the Web of Linked Data is based on the RDF data model and SPARQL is a query language for RDF data. In this paper we challenge this assumption.

* This report presents an extended version of a paper published in ESWC 2012 [11]. The extended version contains proofs for all technical results in the paper (cf. Appendix C).

Contributions In this paper we understand queries as functions over the Web of Linked Data as a whole. To analyze the suitability of SPARQL as a language for such queries, we have to adjust the semantics of SPARQL. More precisely, we have to redefine the scope for evaluating SPARQL algebra expressions. In this paper we discuss two approaches for such an adjustment. The first approach uses a semantics where the scope of a query is the complete set of Linked Data on the Web. We call this semantics *full-Web semantics*. The second approach introduces a family of *reachability-based semantics* which restrict the scope to data that is reachable by traversing certain data links. We emphasize that both approaches allow for query results that are based on data from initially unknown sources and, thus, enable applications to tap the full potential of the Web. Nevertheless, both approaches precisely define the (expected) result for any query.

As a prerequisite for defining the aforementioned semantics and for studying theoretical properties of queries under these semantics, we introduce a theoretical framework. The basis of this framework is a data model that captures the idea of a Web of Linked Data. We model such a Web as an infinite structure of documents that contain RDF data and that are interlinked via this data. Our model allows for infiniteness because the number of entities described in a Web of Linked Data may be infinite; so may the number of documents. The following example illustrates such a case:

Example 1. Let u_i denote an HTTP scheme based URI that identifies the natural number i. There is a countably infinite number of such URIs. The WWW server which is responsible for these URIs may be set up to provide a document for each natural number. These documents may be generated upon request and may contain RDF data including the RDF triple $(u_i, \text{http://.../next}, u_{i+1})$. This triple associates the natural number i with its successor i+1 and, thus, links to the data about i+1 [19]. An example for such a server is provided by the Linked Open Numbers project[1].

In addition to the data model our theoretical framework comprises a computation model. This model is based on a particular type of Turing machine which formally captures the limited data access capabilities of computations over the Web.

We summarize the main contributions of this paper as follows:

- We present a data model and a computation model that provide a theoretical framework to define and to study query languages for the Web of Linked Data.
- We introduce a full-Web semantics and a family of reachability-based semantics for a (hypothetical) use of SPARQL as a language for queries over Linked Data.
- We systematically analyze SPARQL queries under the semantics that we introduce. This analysis includes a discussion of satisfiability, monotonicity, and computability of queries under the different semantics, a comparison of the semantics, and a study of the implications of querying a Web of Linked Data that is infinite.

Related Work Since its emergence the WWW has attracted research on declarative query languages for the Web. For an overview on early work in this area we refer to [8]. Most of this work understands the WWW as a hypertext Web. Nonetheless, some of the foundational work can be adopted for research on Linked Data. The computation model that we use in this paper is an adaptation of the ideas presented in [1] and [15].

[1] http://km.aifb.kit.edu/projects/numbers/

In addition to the early work on Web queries, query execution over Linked Data on the WWW has attracted much attention recently [9,10,12,13,14]. However, existing work primarily focuses on various aspects of (query-local) data management, query execution, and optimization. The only work we are aware of that aims to formally capture the concept of Linked Data and to provide a well-defined semantics for queries in this context is Bouquet et al.'s [7]. They define three types of query methods for conjunctive queries: a bounded method which only uses RDF data referred to in queries, a direct access method which assumes an oracle that provides all RDF graphs which are "relevant" for a given query, and a navigational method which corresponds to a particular reachability-based semantics. For the latter Bouquet et al. define a notion of reachability that allows a query execution system to follow *all* data links. As a consequence, the semantics of queries using this navigational method is equivalent to, what we call, c_{All}-semantics (cf. Section 5.1); it is the most general of our reachability-based semantics. Bouquet et al.'s navigational query model does not support other, more restrictive notions of reachability, as is possible with our model. Furthermore, Bouquet et al. do not discuss full SPARQL, theoretical properties of queries, or the infiniteness of the WWW.

While we focus on the query language SPARQL in the context of Linked Data on the Web, the theoretical properties of SPARQL as a query language for a fixed, predefined collection of RDF data are well understood today [2,3,16,18]. Particularly interesting in our context are semantical equivalences between SPARQL expressions [18] because these equivalences may also be used for optimizing SPARQL queries over Linked Data.

Structure of the paper The remainder of this paper is organized as follows. Section 2 introduces the preliminaries for our work. In Section 3 we present the data model and the computation model. Sections 4 and 5 discuss the full-Web semantics and the reachability-based semantics for SPARQL, respectively. We conclude the paper in Section 6. For full technical proofs of all results in this paper we refer to Appendix C.

2 Preliminaries

This section provides a brief introduction of RDF and the query language SPARQL.

We assume pairwise disjoint, countably infinite sets $\mathcal{U}$ (all HTTP scheme based URIs[2]), $\mathcal{B}$ (blank nodes), $\mathcal{L}$ (literals), and $\mathcal{V}$ (variables, denoted by a leading '?' symbol). An RDF triple t is a tuple $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. For any RDF triple $t = (s, p, o)$ we define $\mathrm{terms}(t) = \{s, p, o\}$ and $\mathrm{uris}(t) = \mathrm{terms}(t) \cap \mathcal{U}$. Overloading function terms, we write $\mathrm{terms}(G) = \bigcup_{t \in G} \mathrm{terms}(t)$ for any (potentially infinite) set G of RDF triples. In contrast to the usual formalization of RDF we allow for infinite sets of RDF triples which we require to study infinite Webs of Linked Data.

In this paper we focus on the core fragment of SPARQL discussed by Pérez et al. [16] and we adopt their formalization approach, that is, we use the algebraic syntax and the compositional set semantics introduced in [16]. *SPARQL expression*s are defined recursively: i) A *triple pattern* $(s, p, o) \in (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L})$ is

[2] For the sake of simplicity we assume in this paper that URIs are HTTP scheme based URIs. However, our models and result may be extended easily for all possible types of URIs.

a SPARQL expression[3]. ii) If P_1 and P_2 are SPARQL expressions, then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ FILTER } R)$ are SPARQL expressions where R is a filter condition. For a formal definition of filter conditions we refer to [16]. To denote the set of all variables in all triple patterns of a SPARQL expression P we write $\text{vars}(P)$.

To define the semantics of SPARQL we introduce *valuations*, that are, partial mappings $\mu : \mathcal{V} \to \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. The *evaluation* of a SPARQL expression P over a potentially infinite set G of RDF triples, denoted by $[\![P]\!]_G$, is a set of valuations. In contrast to the usual case, this set may be infinite in our scenario. The evaluation function $[\![\cdot]\!]_\cdot$ is defined recursively over the structure of SPARQL expressions. Due to space limitations, we do not reproduce the full formal definition of $[\![\cdot]\!]_\cdot$ here. Instead, we refer the reader to the definitions given by Pérez et al. [16]; even if Pérez et al. define $[\![\cdot]\!]_\cdot$ for finite sets of RDF triples, it is trivial to extend their formalism for infiniteness (cf. Appendix B).

A SPARQL expression P is *monotonic* if for any pair G_1, G_2 of (potentially infinite) sets of RDF triples such that $G_1 \subseteq G_2$, it holds that $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. A SPARQL expression P is *satisfiable* if there exists a (potentially infinite) set G of RDF triples such that $[\![P]\!]_G \neq \varnothing$. It is trivial to show that any non-satisfiable expression is monotonic.

In addition to the traditional notion of satisfiability we shall need a more restrictive notion for the discussion in this paper: A SPARQL expression P is *nontrivially satisfiable* if there exists a (potentially infinite) set G of RDF triples and a valuation μ such that i) $\mu \in [\![P]\!]_G$ and ii) μ provides a binding for at least one variable; i.e. $\text{dom}(\mu) \neq \varnothing$.

Example 2. Let $P_{\text{Ex2}} = tp$ be a SPARQL expression that consists of a single triple pattern $tp = (u_1, u_2, u_3)$ where $u_1, u_2, u_3 \in \mathcal{U}$; hence, tp actually is an RDF triple. For any set G of RDF triples for which $(u_1, u_2, u_3) \in G$ it is easy to see that the evaluation of P_{Ex2} over G contains a single, empty valuation $\mu_\varnothing$, that is, $[\![P_{\text{Ex2}}]\!]_G = \{\mu_\varnothing\}$ where $\text{dom}(\mu_\varnothing) = \varnothing$. In contrast, for any other set G of RDF triples it holds $[\![P_{\text{Ex2}}]\!]_G = \varnothing$. Hence, P_{Ex2} is *not* nontrivially satisfiable (although it is satisfiable).

3 Modeling a Web of Linked Data

In this section we introduce theoretical foundations which shall allow us to define and to analyze query models for Linked Data. In particular, we propose a data model and introduce a computation model. For these models we assume a *static view* of the Web; that is, no changes are made to the data on the Web during the execution of a query.

3.1 Data Model

We model the Web of Linked Data as a potentially infinite structure of interlinked documents. Such documents, which we call Linked Data documents, or *LD document*s for short, are accessed via URIs and contain data that is represented as a set of RDF triples.

Definition 1. *Let $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ be the infinite set of all possible RDF triples. A* ***Web of Linked Data*** *is a tuple $W = (D, data, adoc)$ where:*

[3] For the sake of a more straightforward formalization we do not permit blank nodes in triple patterns. In practice, each blank node in a SPARQL query can be replaced by a new variable.

- D *is a (finite or countably infinite) set of symbols that represent LD documents.*
- $data$ *is a total mapping* $data : D \to 2^{\mathcal{T}}$ *such that* $\forall\, d \in D : data(d)$ *is finite and* $\forall d_1, d_2 \in D : d_1 \neq d_2 \Rightarrow \text{terms}\big(data(d_1)\big) \cap \mathcal{B} \neq \text{terms}\big(data(d_2)\big) \cap \mathcal{B}$.
- $adoc$ *is a partial, surjective mapping* $adoc \colon \mathcal{U} \to D$.

While the three elements D, $data$, and $adoc$ completely define a Web of Linked Data in our model, we point out that these elements are abstract concepts and, thus, are not available to a query execution system. However, by retrieving LD documents, such a system may gradually obtain information about the Web. Based on this information the system may (partially) materialize these three elements. In the following we discuss the three elements and introduce additional concepts that we need to define queries.

We say a Web of Linked Data $W = (D, data, adoc)$ is *finite* if and only if D is finite; otherwise, W is *infinite*. Our model allows for infiniteness to cover cases where Linked Data about an infinite number of identifiable entities is generated on the fly. The Linked Open Numbers project (cf. Example 1) illustrates that such cases are possible in practice. Another example is the LinkedGeoData project[4] which provides Linked Data about any circular and rectangular area on Earth [4]. Covering these cases enables us to model queries over such data and analyze the effects of executing such queries.

Even if a Web of Linked Data $W = (D, data, adoc)$ is infinite, Definition 1 requires countability for D. We emphasize that this requirement does not restrict us in modeling the WWW as a Web of Linked Data: In the WWW we use URIs to locate documents that contain Linked Data. Even if URIs are not limited in length, they are words over a finite alphabet. Thus, the infinite set of all possible URIs is countable, as is the set of all documents that may be retrieved using URIs.

The mapping $data$ associates each LD document $d \in D$ in a Web of Linked Data $W = (D, data, adoc)$ with a finite set of RDF triples. In practice, these triples are obtained by parsing d after d has been retrieved from the Web. The actual retrieval mechanism is not relevant for our model. However, as prescribed by the RDF data model, Definition 1 requires that the data of each $d \in D$ uses a unique set of blank nodes.

To denote the (potentially infinite but countable) set of *all RDF triples* in W we write $\text{AllData}(W)$; i.e. it holds: $\text{AllData}(W) = \big\{data(d) \,|\, d \in D\big\}$.

Since we use URIs as identifiers for entities, we say that an LD document $d \in D$ *describes* the entity identified by URI $u \in \mathcal{U}$ if there exists $(s, p, o) \in data(d)$ such that $s = u$ or $o = u$. Notice, there might be multiple LD documents that describe an entity identified by u. However, according to the Linked Data principles, each $u \in \mathcal{U}$ may also serve as a reference to a specific LD document which is considered as an authoritative source of data about the entity identified by u. We model the relationship between URIs and authoritative LD documents by mapping $adoc$. Since some LD documents may be authoritative for multiple entities, we do not require injectivity for $adoc$. The "real world" mechanism for dereferencing URIs (i.e. learning about the location of the authoritative LD document) is not relevant for our model. For each $u \in \mathcal{U}$ that cannot be dereferenced (i.e. "broken links") or that is not used in W it holds $u \notin \text{dom}(adoc)$.

A URI $u \in \mathcal{U}$ with $u \in \text{dom}(adoc)$ that is used in the data of an LD document $d_1 \in D$ constitutes a *data link* to the LD document $d_2 = adoc(u) \in D$. These data links form a

[4] http://linkedgeodata.org

graph structure which we call *link graph*. The vertices in such a graph represent the LD documents of the corresponding Web of Linked Data; edges represent data links.

To study the monotonicity of queries over a Web of Linked Data we require a concept of containment for such Webs. For this purpose, we introduce the notion of an induced subweb which resembles the concept of induced subgraphs in graph theory.

Definition 2. *Let $W = (D, data, adoc)$ and $W' = (D', data', adoc')$ be Webs of Linked Data. W' is an **induced subweb of** W if i) $D' \subseteq D$, ii) $\forall\, d \in D' : data'(d) = data(d)$, and iii) $\forall\, u \in \mathcal{U}_{D'} : adoc'(u) = adoc(u)$ where $\mathcal{U}_{D'} = \{u \in \mathcal{U} \mid adoc(u) \in D'\}$.*

It can be easily seen from Definition 2 that specifying D' is sufficient to unambiguously define an induced subweb $(D', data', adoc')$ of a given Web of Linked Data. Furthermore, it is easy to verify that for an induced subweb W' of a Web of Linked Data W it holds $\mathrm{AllData}(W') \subseteq \mathrm{AllData}(W)$.

In addition to the structural part, our data model introduces a general understanding of queries over a Web of Linked Data:

Definition 3. *Let $\mathcal{W}$ be the infinite set of all possible Webs of Linked Data (i.e. all 3-tuples that correspond to Definition 1) and let Ω be the infinite set of all possible valuations. A **Linked Data query** q is a total function $q\colon \mathcal{W} \to 2^{\Omega}$.*

The notions of satisfiability and monotonicity carry over naturally to Linked Data queries: A Linked Data query q is *satisfiable* if there exists a Web of Linked Data W such that $q(W)$ is not empty. A Linked Data query q is *nontrivially satisfiable* if there exists a Web of Linked Data W and a valuation μ such that i) $\mu \in q(W)$ and ii) $\mathrm{dom}(\mu) \neq \varnothing$. A Linked Data query q is *monotonic* if for every pair W_1, W_2 of Webs of Linked Data it holds: If W_1 is an induced subweb of W_2, then $q(W_1) \subseteq q(W_2)$.

3.2 Computation Model

Usually, functions are computed over structures that are assumed to be fully (and directly) accessible. In contrast, we focus on Webs of Linked Data in which accessibility is limited: To discover LD documents and access their data we have to dereference URIs, but the full set of those URIs for which we may retrieve documents is unknown. Hence, to properly analyze a query model for Webs of Linked Data we must define a model for computing functions on such a Web. This section introduces such a model.

In the context of queries over a hypertext-centric view of the WWW, Abiteboul and Vianu introduce a specific Turing machine called Web machine [1]. Mendelzon and Milo propose a similar machine model [15]. These machines formally capture the limited data access capabilities on the WWW and thus present an adequate abstraction for computations over a structure such as the WWW. Based on these machines the authors introduce particular notions of computability for queries over the WWW. These notions are: *(finitely) computable queries*, which correspond to the traditional notion of computability; and *eventually computable queries* whose computation may not terminate but each element of the query result will eventually be reported during the computation. We adopt the ideas of Abiteboul and Vianu and of Mendelzon and Milo for our work. More precisely, we adapt the idea of a Web machine to our scenario of a Web of Linked

Data. We call our machine a *Linked Data machine* (or LD machine, for short). Based on this machine we shall define finite and eventual computability for Linked Data queries.

Encoding (fragments of) a Web of Linked Data $W = (D, data, adoc)$ on the tapes of such an LD machine is straightforward because all relevant structures, such as the sets D or $\mathcal{U}$, are countably infinite. In the remainder of this paper we write $\mathrm{enc}(x)$ to denote the encoding of some element x (e.g. a single RDF triple, a set of triples, a full Web of Linked Data, a valuation, etc.). For a detailed definition of the encodings we use in this paper, we refer to Appendix A. We now define LD machine:

Definition 4. *An **LD machine** is a multi-tape Turing machine with five tapes and a finite set of states, including a special state called* expand*. The five tapes include two, read-only input tapes: i) an ordinary input tape and ii) a right-infinite* Web tape *which can only be accessed in the expand state; two work tapes: iii) an ordinary, two-way infinite work tape and iv) a right-infinite* link traversal tape*; and v) a right-infinite, append-only output tape. Initially, the work tapes and the output tape are empty, the Web tape contains a (potentially infinite) word that encodes a Web of Linked Data, and the ordinary input tape contains an encoding of further input (if any). Any LD machine operates like an ordinary multi-tape Turing machine except when it reaches the expand state. In this case LD machines perform the following* expand procedure*: The machine inspects the word currently stored on the link traversal tape. If the suffix of this word is the encoding $\mathrm{enc}(u)$ of some URI $u \in \mathcal{U}$ and the word on the Web tape contains $\sharp\,\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$, then the machine appends $\mathrm{enc}(adoc(u))\,\sharp$ to the (right) end of the word on the link traversal tape by copying from the Web tape; otherwise, the machine appends $\sharp$ to the word on the link traversal tape.*

Notice how any LD machine M is limited in the way it may access a Web of Linked Data $W = (D, data, adoc)$ that is encoded on its Web tape: M may use the data of any particular $d \in D$ only after it performed the expand procedure using a URI $u \in \mathcal{U}$ for which $adoc(u) = d$. Hence, the expand procedure simulates a URI based lookup which conforms to the (typical) data access method on the WWW. We now use LD machines to adapt the notion of finite and eventual computability [1] for Linked Data queries:

Definition 5. *A Linked Data query q is **finitely computable** if there exists an LD machine which, for any Web of Linked Data W encoded on the Web tape, halts after a finite number of steps and produces a possible encoding of $q(W)$ on its output tape.*

Definition 6. *A Linked Data q query is **eventually computable** if there exists an LD machine whose computation on any Web of Linked Data W encoded on the Web tape has the following two properties: 1.) the word on the output tape at each step of the computation is a prefix of a possible encoding of $q(W)$ and 2.) the encoding $\mathrm{enc}(\mu')$ of any $\mu' \in q(W)$ becomes part of the word on the output tape after a finite number of computation steps.*

Any machine for a non-satisfiable query may immediately report the empty result. Thus:

Fact 1. *Non-satisfiable Linked Data queries are finitely computable.*

In our analysis of SPARQL-based Linked Data queries we shall discuss decision problems that have a Web of Linked Data W as input. For such problems we assume the computation may only be performed by an LD machine with $\mathrm{enc}(W)$ on its Web tape:

Definition 7. *Let $\mathcal{W}'$ be a (potentially infinite) set of Webs of Linked Data (each of which may be infinite itself); let $\mathcal{X}$ be an arbitrary (potentially infinite) set of finite structures; and let $DP \subseteq \mathcal{W}' \times \mathcal{X}$. The decision problem for DP, that is, decide for any $(W, X) \in \mathcal{W}' \times \mathcal{X}$ whether $(W, X) \in DP$, is **LD machine decidable** if there exist an LD machine whose computation on any $W \in \mathcal{W}'$ encoded on the Web tape and any $X \in \mathcal{X}$ encoded on the ordinary input tape, has the following property: The machine halts in an accepting state if $(W, X) \in DP$; otherwise the machine halts in a rejecting state.*

Obviously, any (Turing) decidable problem that does not have a Web of Linked Data as input, is also LD machine decidable because LD machines are Turing machines; for these problems the corresponding set $\mathcal{W}'$ is empty .

4 Full-Web Semantics

Based on the concepts introduced in the previous section we now define and study approaches that adapt SPARQL as a language for expressing Linked Data queries.

The first approach that we discuss is full-Web semantics where the scope of each query is the complete set of Linked Data on the Web. Hereafter, we refer to SPARQL queries under this full-Web semantics as SPARQL$_{\mathsf{LD}}$ queries. The definition of these queries is straightforward and makes use of SPARQL expressions and their semantics:

Definition 8. *Let P be a SPARQL expression. The **SPARQL$_{\mathsf{LD}}$ query that uses** P, denoted by $\mathcal{Q}^P$, is a Linked Data query that, for any Web of Linked Data W, is defined as: $\mathcal{Q}^P(W) = [\![P]\!]_{\mathrm{AllData}(W)}$. Each valuation $\mu \in \mathcal{Q}^P(W)$ is a **solution** for $\mathcal{Q}^P$ in W.*

In the following we study satisfiability, monotonicity, and computability of SPARQL$_{\mathsf{LD}}$ queries and we discuss implications of querying Webs of Linked Data that are infinite.

4.1 Satisfiability, Nontrivial Satisfiability, Monotonicity, and Computability

For satisfiability and monotonicity we may show the following dependencies.

Proposition 1. *Let $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P.*

1. *$\mathcal{Q}^P$ is satisfiable if and only if P is satisfiable.*
2. *$\mathcal{Q}^P$ is nontrivially satisfiable if and only if P is nontrivially satisfiable.*
3. *$\mathcal{Q}^P$ is monotonic if and only if P is monotonic.*

We now discuss computability. Since all non-satisfiable SPARQL$_{\mathsf{LD}}$ queries are finitely computable (recall Fact 1), we focus on satisfiable SPARQL$_{\mathsf{LD}}$ queries. Our first main result shows that the computability of such queries depends on their monotonicity:

Theorem 1. *If a satisfiable SPARQL$_{\mathsf{LD}}$ query is monotonic, then it is eventually computable (but not finitely computable); otherwise, it is not even eventually computable.*

In addition to a direct dependency between monotonicity and computability, Theorem 1 shows that not any satisfiable SPARQL$_{\mathsf{LD}}$ query is finitely computable; instead, such queries are at best eventually computable. The reason for this limitation is the infiniteness of $\mathcal{U}$: To (fully) compute a satisfiable SPARQL$_{\mathsf{LD}}$ query, an LD machine requires

access to the data of *all* LD documents in the queried Web of Linked Data. Recall that, initially, the machine has no information about what URI to use for performing an expand procedure with which it may access any particular document. Hence, to ensure that all documents have been accessed, the machine must expand all $u \in \mathcal{U}$. This process never terminates because $\mathcal{U}$ is infinite. Notice, a real query system for the WWW would have a similar problem: To guarantee that such a system sees all documents, it must enumerate and lookup all (HTTP scheme) URIs.

The computability of any Linked Data query is a general, input independent property which covers the worst case (recall, the requirements given in Definitions 5 and 6 must hold for any Web of Linked Data). As a consequence, in certain cases the computation of some (eventually computable) SPARQL$_{\mathsf{LD}}$ queries may still terminate:

Example 3. Let $\mathcal{Q}^{P_{\mathsf{Ex}2}}$ be a monotonic SPARQL$_{\mathsf{LD}}$ query which uses the SPARQL expression $P_{\mathsf{Ex}2} = (u_1, u_2, u_3)$ that we introduce in Example 2. Recall, $P_{\mathsf{Ex}2}$ is satisfiable but *not* nontrivially satisfiable. The same holds for $\mathcal{Q}^{P_{\mathsf{Ex}2}}$ (cf. Proposition 1). An LD machine for $\mathcal{Q}^{P_{\mathsf{Ex}2}}$ may take advantage of this fact: As soon as the machine discovers an LD document which contains RDF triple (u_1, u_2, u_3), the machine may halt (after reporting $\{\mu_\varnothing\}$ with $\mathrm{dom}(\mu_\varnothing) = \varnothing$ as the complete query result). In this particular case the machine would satisfy the requirements for finite computability. However, $\mathcal{Q}^{P_{\mathsf{Ex}2}}$ is still only eventually computable because there exist Webs of Linked Data that do not contain any LD document with RDF triple (u_1, u_2, u_3); any (complete) LD machine based computation of $\mathcal{Q}^{P_{\mathsf{Ex}2}}$ over such a Web cannot halt (cf. proof of Theorem 1).

The example illustrates that the computation of an eventually computable query over a particular Web of Linked Data may terminate. This observation leads us to a decision problem which we denote as TERMINATION(SPARQL$_{\mathsf{LD}}$). This problem takes a Web of Linked Data W and a satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ as input and asks whether an LD machine exists that computes $\mathcal{Q}^P(W)$ and halts. For discussing this problem we note that the query in Example 3 represents a special case, that is, SPARQL$_{\mathsf{LD}}$ queries which are satisfiable but not nontrivially satisfiable. The reason why an LD machine for such a query may halt, is the implicit knowledge that the query result is complete once the machine identified the empty valuation $\mu_\varnothing$ as a solution. Such a completeness criterion does not exist for any nontrivially satisfiable SPARQL$_{\mathsf{LD}}$ query:

Lemma 1. *There is not any nontrivially satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ for which exists an LD machine that, for any Web of Linked Data W encoded on the Web tape, halts after a finite number of computation steps and outputs an encoding of $\mathcal{Q}^P(W)$.*

Lemma 1 shows that the answer to TERMINATION(SPARQL$_{\mathsf{LD}}$) is negative in most cases. However, the problem in general is undecidable (for LD machines) since the input for the problem includes queries that correspond to the aforementioned special case.

Theorem 2. TERMINATION(SPARQL$_{\mathsf{LD}}$) *is not LD machine decidable.*

4.2 Querying an Infinite Web of Linked Data

The limited computability of SPARQL$_{\mathsf{LD}}$ queries that our results in the previous section show, is a consequence of the infiniteness of $\mathcal{U}$ and not of a possible infiniteness of the

queried Web. We now focus on the implications of potentially infinite Webs of Linked Data for SPARQL$_{\mathsf{LD}}$ queries. However, we assume a *finite* Web first:

Proposition 2. *SPARQL$_{LD}$ queries over a finite Web of Linked Data have a finite result.*

The following example illustrates that a similarly general statement does not exist when the queried Web is infinite such as the WWW.

Example 4. Let $W_{\mathsf{inf}} = (D_{\mathsf{inf}}, data_{\mathsf{inf}}, adoc_{\mathsf{inf}})$ be an *infinite* Web of Linked Data that contains LD documents for all natural numbers (similar to the documents in Example 1). Hence, for each natural number[5] $k \in \mathbb{N}^+$, identified by $u_k \in \mathcal{U}$, exists an LD document $adoc_{\mathsf{inf}}(u_k) = d_k \in D_{\mathsf{inf}}$ such that $data_{\mathsf{inf}}(d_k) = \{(u_k, \mathsf{succ}, u_{k+1})\}$ where $\mathsf{succ} \in \mathcal{U}$ identifies the successor relation for $\mathbb{N}^+$. Furthermore, let $P_1 = (u_1, \mathsf{succ}, ?v)$ and $P_2 = (?x, \mathsf{succ}, ?y)$ be SPARQL expressions. It can be seen easily that the result of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^{P_1}$ over W_{inf} is finite, whereas, $\mathcal{Q}^{P_2}(W_{\mathsf{inf}})$ is infinite.

The example demonstrates that some SPARQL$_{\mathsf{LD}}$ queries have a finite result over some infinite Web of Linked Data and some queries have an infinite result. Consequently, we are interested in a decision problem FINITENESS(SPARQL$_{\mathsf{LD}}$) which asks, given a (potentially infinite) Web of Linked Data W and a satisfiable SPARQL expression P, whether $\mathcal{Q}^P(W)$ is finite. Unfortunately, we cannot answer the problem in general:

Theorem 3. FINITENESS(SPARQL$_{\mathsf{LD}}$) *is not LD machine decidable.*

5 Reachability-Based Semantics

Our results in the previous section show that SPARQL queries under full-Web semantics have a very limited computability. As a consequence, any SPARQL-based query approach for Linked Data that uses full-Web semantics requires some ad hoc mechanism to abort query executions and, thus, has to accept incomplete query results. Depending on the abort mechanism the query execution may even be nondeterministic. If we take these issues as an obstacle, we are interested in an alternative, well-defined semantics for SPARQL over Linked Data. In this section we discuss a family of such semantics which we call *reachability-based semantics*. These semantics restrict the scope of queries to data that is reachable by traversing certain data links using a given set of URIs as starting points. Hereafter, we refer to queries under any reachability-based semantics as *SPARQL$_{LD(R)}$ queries*. In the remainder of this section we formally introduce reachability-based semantics, discuss theoretical properties of SPARQL$_{\mathsf{LD(R)}}$ queries, and compare SPARQL$_{\mathsf{LD(R)}}$ to SPARQL$_{\mathsf{LD}}$.

5.1 Definition

The basis of any reachability-based semantics is a notion of reachability of LD documents. Informally, an LD document is reachable if there exists a (specific) path in the

[5] In this paper we write $\mathbb{N}^+$ to denote the set of all natural numbers without zero.

link graph of a Web of Linked Data to the document in question; the potential starting points for such a path are LD documents that are authoritative for a given set of entities. However, allowing for arbitrary paths might be questionable in practice because this approach would require following *all* data links (recursively) for answering a query completely. Consequently, we introduce the notion of a reachability criterion that supports an explicit specification of what data links should be followed.

Definition 9. *Let $\mathcal{T}$ be the infinite set of all possible RDF triples and let $\mathcal{P}$ be the infinite set of all possible SPARQL expressions. A **reachability criterion** c is a (Turing) computable function $c : \mathcal{T} \times \mathcal{U} \times \mathcal{P} \to \{\text{true}, \text{false}\}$.*

An example for a reachability criterion is c_{All} which corresponds to the aforementioned approach of allowing for arbitrary paths to reach LD documents; hence, for each tuple $(t, u, Q) \in \mathcal{T} \times \mathcal{U} \times \mathcal{Q}$ it holds $c_{\mathsf{All}}(t, u, Q) = \text{true}$. The complement of c_{All} is c_{None} which *always* returns false. Another example is c_{Match} which specifies the notion of reachability that we use for link traversal based query execution [10,12].

$$c_{\mathsf{Match}}\Big(t, u, P\Big) = \begin{cases} \text{true} & \text{if there exists a triple pattern } tp \text{ in } P \text{ and } t \text{ matches } tp, \\ \text{false} & \text{else.} \end{cases}$$

where an RDF triple $t = (x_1, x_2, x_3)$ *matches* a triple pattern $tp = (\tilde{x_1}, \tilde{x_2}, \tilde{x_3})$ if for all $i \in \{1, 2, 3\}$ holds: If $\tilde{x_i} \notin \mathcal{V}$, then $\tilde{x_i} = x_i$.

We call a reachability criterion c_1 *less restrictive than* another criterion c_2 if i) for each $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ for which $c_2(t, u, P) = \text{true}$, also holds $c_1(t, u, P) = \text{true}$ and ii) there exist a $(t', u', P') \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ such that $c_1(t', u', P') = \text{true}$ but $c_2(t', u', P') = \text{false}$. It can be seen that c_{All} is the least restrictive criterion, whereas c_{None} is the most restrictive criterion. We now define reachability of LD documents:

Definition 10. *Let $S \subset \mathcal{U}$ be a finite set of seed URIs; let c be a reachability criterion; let P be a SPARQL expression; and let $W = (D, data, adoc)$ be a Web of Linked Data. An LD document $d \in D$ **is** (c, P)**-reachable from** S **in** W if either*

1. *there exists a URI $u \in S$ such that $adoc(u) = d$; or*
2. *there exist $d' \in D$, $t \in data(d')$, and $u \in \text{uris}(t)$ such that i) d' is (c, P)-reachable from S in W, ii) $adoc(u) = d$, and iii) $c(t, u, P) = \text{true}$.*

Based on reachability of LD documents we define reachable parts of a Web of Linked Data. Such a part is an induced subweb covering all reachable LD documents. Formally:

Definition 11. *Let $S \subset \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; let P be a SPARQL expression; and let $W = (D, data, adoc)$ be a Web of Linked Data. The (S, c, P)**-reachable part of** W, denoted by $W_c^{(S,P)}$, is an induced subweb $(D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ of W such that $D_{\mathfrak{R}} = \big\{d \in D \mid d \text{ is } (c, P)\text{-reachable from } S \text{ in } W\big\}$.*

We now use the concept of reachable parts to define $\text{SPARQL}_{\text{LD(R)}}$ queries.

Definition 12. *Let $S \subset \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; and let P be a SPARQL expression. The **SPARQL$_{\text{LD(R)}}$ query that uses** P**,** S**, and** c, denoted by $\mathcal{Q}_c^{P,S}$, is a Linked Data query that, for any Web of Linked Data W, is defined as $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\text{AllData}(W_c^{(S,P)})}$ (where $W_c^{(S,P)}$ is the (S, c, P)-reachable part of W).*

As can be seen from Definition 12, our notion of SPARQL$_{\text{LD(R)}}$ consists of a family of (reachability-based) query semantics, each of which is characterized by a certain reachability criterion. Therefore, we refer to SPARQL$_{\text{LD(R)}}$ queries for which we use a particular reachability criterion c as SPARQL$_{\text{LD(R)}}$ queries *under c-semantics*.

Definition 12 also shows that query results depend on the given set $S \subset \mathcal{U}$ of seed URIs. It is easy to see that any SPARQL$_{\text{LD(R)}}$ query which uses an empty set of seed URIs is not satisfiable and, thus, monotonic and finitely computable. We therefore consider only nonempty sets of seed URIs in the remainder of this paper.

5.2 Completeness and Infiniteness

Definition 12 defines precisely what the sound and complete result of any SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ over any Web of Linked Data W is. However, in contrast to SPARQL$_{\text{LD}}$, it is not guaranteed that such a (complete) SPARQL$_{\text{LD(R)}}$ result is complete w.r.t. all data on W. This difference can be attributed to the fact that the corresponding (S, c, P)-reachable part of W may not cover W as a whole. We emphasize that such an incomplete coverage is even possible for the reachability criterion c_{All} because the link graph of W may not be connected; therefore, c_{All}-semantics differs from full-Web semantics. The following result relates SPARQL$_{\text{LD(R)}}$ queries to their SPARQL$_{\text{LD}}$ counterparts.

Proposition 3. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\text{LD(R)}}$ query; let $\mathcal{Q}^P$ be the SPARQL$_{\text{LD}}$ query that uses the same SPARQL expression as $\mathcal{Q}_c^{P,S}$; let W be a Web of Linked Data. It holds:*

1. *If $\mathcal{Q}^P$ is monotonic, then $\mathcal{Q}_c^{P,S}(W) \subseteq \mathcal{Q}^P(W)$.*
2. *$\mathcal{Q}_c^{P,S}(W) = \mathcal{Q}^P(W_c^{(S,P)})$. (recall, $W_c^{(S,P)}$ is the (S, c, P)-reachable part of W)*

Since any SPARQL$_{\text{LD}}$ query over a finite Web of Linked Data has a finite result (cf. Proposition 2), we use Proposition 3, case 2, to show the same for SPARQL$_{\text{LD(R)}}$:

Proposition 4. *The result of any SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ over a finite Web of Linked Data W is finite; so is the (S, c, P)-reachable part of W.*

For the case of an *infinite* Web of Linked Data the results of SPARQL$_{\text{LD(R)}}$ queries may be either finite or infinite. In Example 4 we found the same heterogeneity for SPARQL$_{\text{LD}}$. However, for SPARQL$_{\text{LD(R)}}$ we may identify the following dependencies.

Proposition 5. *Let $S \subset \mathcal{U}$ be a finite, nonempty set of URIs; let c and c' be reachability criteria; and let P be a SPARQL expression. Let W be an* infinite *Web of Linked Data.*

1. *$W_{c_{\text{None}}}^{(S,P)}$ is always finite; so is $\mathcal{Q}_{c_{\text{None}}}^{P,S}(W)$.*
2. *If $W_c^{(S,P)}$ is finite, then $\mathcal{Q}_c^{P,S}(W)$ is finite.*
3. *If $\mathcal{Q}_c^{P,S}(W)$ is infinite, then $W_c^{(S,P)}$ is infinite.*
4. *If c is less restrictive than c' and $W_c^{(S,P)}$ is finite, then $W_{c'}^{(S,P)}$ is finite.*
5. *If c' is less restrictive than c and $W_c^{(S,P)}$ is infinite, then $W_{c'}^{(S,P)}$ is infinite.*

Proposition 5 provides valuable insight into the dependencies between reachability criteria, the (in)finiteness of reachable parts of an infinite Web, and the (in)finiteness of query results. In practice, however, we are primarily interested in answering two decision problems: FINITENESSREACHABLEPART and FINITENESS(SPARQL$_{\text{LD(R)}}$).

While the latter problem is the SPARQL$_{\mathsf{LD(R)}}$ equivalent to FINITENESS(SPARQL$_{\mathsf{LD}}$) (cf. Section 4.2), the former has the same input as FINITENESS(SPARQL$_{\mathsf{LD(R)}}$) (that is, a Web of Linked Data and a SPARQL$_{\mathsf{LD(R)}}$ query) and asks whether the corresponding reachable part of the given Web is finite. Both problems are undecidable in our context:

Theorem 4. FINITENESSREACHABLEPART *and* FINITENESS(SPARQL$_{\mathsf{LD(R)}}$) *are not LD machine decidable.*

5.3 Satisfiability, Nontrivial Satisfiability, Monotonicity, and Computability

We now investigate satisfiability, nontrivial satisfiability, monotonicity, and computability of SPARQL$_{\mathsf{LD(R)}}$ queries. First, we identify the following dependencies.

Proposition 6. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses a nonempty $S \subset \mathcal{U}$.*

1. *$\mathcal{Q}_c^{P,S}$ is satisfiable if and only if P is satisfiable.*
2. *$\mathcal{Q}_c^{P,S}$ is nontrivially satisfiable if and only if P is nontrivially satisfiable.*
3. *$\mathcal{Q}_c^{P,S}$ is monotonic if P is monotonic.*

Proposition 6 reveals a first major difference between SPARQL$_{\mathsf{LD(R)}}$ and SPARQL$_{\mathsf{LD}}$: The statement about monotonicity in that proposition is only a material conditional, whereas it is a biconditional in the case of SPARQL$_{\mathsf{LD}}$ (cf. Proposition 1). The reason for this disparity are SPARQL$_{\mathsf{LD(R)}}$ queries for which monotonicity is independent of the corresponding SPARQL expression. The following proposition identifies such a case.

Proposition 7. *Any SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}$ is monotonic if $|S| = 1$.*

Before we may come back to the aforementioned disparity, we focus on the computability of SPARQL$_{\mathsf{LD(R)}}$ queries. We first show the following, noteworthy result.

Lemma 2. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that is nontrivially satisfiable. There exists an LD machine that computes $\mathcal{Q}_c^{P,S}$ over any (potentially infinite) Web of Linked Data W and that halts after a finite number of computation steps with an encoding of $\mathcal{Q}_c^{P,S}(W)$ on its output tape if and only if the (S, c, P)-reachable part of W is finite.*

The importance of Lemma 2 lies in showing that some computations of nontrivially satisfiable SPARQL$_{\mathsf{LD(R)}}$ queries may terminate. This possibility presents another major difference between SPARQL$_{\mathsf{LD(R)}}$ and SPARQL$_{\mathsf{LD}}$ (recall Lemma 1 which shows that any possible computation of nontrivially satisfiable SPARQL$_{\mathsf{LD}}$ queries never terminates). Based on Lemma 2 we may even show that a particular class of satisfiable SPARQL$_{\mathsf{LD(R)}}$ queries are finitely computable. This class comprises all queries that use a reachability criterion which ensures the finiteness of reachable parts of any queried Web of Linked Data. We define this property of reachability criteria as follows:

Definition 13. *A reachability criterion c **ensures finiteness** if for any Web of Linked Data W, any (finite) set $S \subset \mathcal{U}$ of seed URIs, and any SPARQL expression P, the (S, c, P)-reachable part of W is finite.*

We may now show the aforementioned result:

Proposition 8. *Let c be a reachability criterion that ensures finiteness. SPARQL$_{LD(R)}$ queries under c-semantics are finitely computable.*

While it remains an open question whether the property to ensure finiteness is decidable for all reachability criteria, it is easy to verify the property for criteria which always only accept a given, constant set of data links. For a formal discussion of such criteria, which we call *constant reachability criteria*, we refer to Appendix D. c_{None} is a special case of these criteria; Proposition 5, case 1, verifies that c_{None} ensures finiteness.

Notice, for any reachability criterion c that ensures finiteness, the computability of SPARQL$_{\mathsf{LD(R)}}$ queries under c-semantics does not depend on the monotonicity of these queries. This independence is another difference to SPARQL$_{\mathsf{LD}}$ queries (recall Theorem 1). However, for any other reachability criterion (including c_{Match} and c_{All}), we have a similar dependency between monotonicity and computability of (satisfiable) SPARQL$_{\mathsf{LD(R)}}$ queries, that we have for SPARQL$_{\mathsf{LD}}$ queries (recall Theorem 1):

Theorem 5. *Let c_{nf} be a reachability criterion that does not ensure finiteness. If a satisfiable SPARQL$_{LD(R)}$ query $\mathcal{Q}^{P,S}_{c_{nf}}$ (under c_{nf}-semantics) is monotonic, then $\mathcal{Q}^{P,S}_{c_{nf}}$ is either finitely computable or eventually computable; otherwise, $\mathcal{Q}^{P,S}_{c_{nf}}$ may not even be eventually computable.*

By comparing Theorems 1 and 5 we notice that SPARQL$_{\mathsf{LD}}$ queries and SPARQL$_{\mathsf{LD(R)}}$ queries (that use a reachability criterion which does not ensure finiteness) feature a similarly limited computability. However, the reasons for both of these results differ significantly: In the case of SPARQL$_{\mathsf{LD}}$ the limitation follows from the infiniteness of $\mathcal{U}$, whereas, for SPARQL$_{\mathsf{LD(R)}}$ the limitation is a consequence of the possibility to query an infinitely large Web of Linked Data.

However, even if the computability of many SPARQL$_{\mathsf{LD(R)}}$ queries is as limited as that of their SPARQL$_{\mathsf{LD}}$ counterparts, there is another major difference: Lemma 2 shows that for (nontrivially satisfiable) SPARQL$_{\mathsf{LD(R)}}$ queries which are not finitely computable, the computation over some Webs of Linked Data may still terminate; this includes all finite Webs (cf. Proposition 4) but also some infinite Webs (cf. proof of Lemma 2). Such a possibility does not exist for nontrivially satisfiable SPARQL$_{\mathsf{LD}}$ queries (cf. Lemma 1). Nonetheless, the termination problem for SPARQL$_{\mathsf{LD(R)}}$ is undecidable in our context.

Theorem 6. TERMINATION(SPARQL$_{\mathsf{LD(R)}}$) *is not LD machine decidable.*

We now come back to the impossibility for showing that SPARQL$_{\mathsf{LD(R)}}$ queries (with a nonempty set of seed URIs) are monotonic *only if* their SPARQL expression is monotonic. Recall, for some SPARQL$_{\mathsf{LD(R)}}$ queries monotonicity is irrelevant for identifying the computability (cf. Proposition 8). We are primarily interested in the monotonicity of all other (satisfiable) SPARQL$_{\mathsf{LD(R)}}$ queries because for those queries computability depends on monotonicity as we show in Theorem 5. Remarkably, for those queries it is possible to show the required dependency that was missing from Proposition 6:

Proposition 9. *Let $\mathcal{Q}^{P,S}_{c_{nf}}$ be a SPARQL$_{LD(R)}$ query that uses a finite, nonempty $S \subset \mathcal{U}$ and a reachability criterion c_{nf} which does not ensure finiteness. $\mathcal{Q}^{P,S}_{c_{nf}}$ is monotonic only if P is monotonic.*

6 Conclusions

Our investigation of SPARQL as a language for Linked Data queries reveals the following main results. Some special cases aside, the computability of queries under any of the studied semantics is limited and no guarantee for termination can be given. For reachability-based semantics it is at least possible that some of the (non-special case) query computations terminate; although, in general it is undecidable which. As a consequence, any SPARQL-based query system for Linked Data on the Web must be prepared for query executions that discover an infinite amount of data and that do not terminate.

Our results also show that –for reachability-based semantics– the aforementioned issues must be attributed to the possibility for infiniteness in the queried Web (which is a result of data generating servers). Therefore, it seems worthwhile to study approaches for detecting whether the execution of a $\text{SPARQL}_{\text{LD(R)}}$ query traverses an infinite path in the queried Web. However, the mentioned issues may also be addressed by another, alternative well-defined semantics that restricts the scope of queries even further (or differently) than our reachability-based semantics. It remains an open question how such an alternative may still allow for queries that tap the full potential of the Web.

We also show that computability depends on satisfiability and monotonicity and that for (almost all) SPARQL-based Linked Data queries, these two properties directly correspond to the same property for the used SPARQL expression. While Arenas and Pérez show that the core fragment of SPARQL without OPT is monotonic [3], it requires further work to identify (non-)satisfiable and (non-)monotonic fragments and, thus, enable an explicit classification of SPARQL-based Linked Data queries w.r.t. computability.

References

1. S. Abiteboul and V. Vianu. Queries and computation on the web. *Theoretical Computer Science*, 239(2), 2000.
2. R. Angles and C. Gutierrez. The expressive power of SPARQL. In *ISWC*, 2008.
3. M. Arenas and J. Pérez. Querying Semantic Web Data with SPARQL. In *PODS*, 2011.
4. S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData – adding a spatial dimension to the web of data. In *Proc. of the 8th Int. Semantic Web Conference (ISWC)*, 2009.
5. T. Berners-Lee. Linked Data. http://www. w3.org/DesignIssues/LinkedData.html, 2006.
6. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – the story so far. *Journal on Semantic Web and Information Systems*, 5(3), 2009.
7. P. Bouquet, C. Ghidini, and L. Serafini. Querying the web of data: A formal approach. In *Proc of the 4th Asian Semantic Web Conference (ASWC)*, 2009.
8. D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3), 1998.
9. A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data Summaries for On-Demand Queries over Linked Data. In *WWW*, 2010.
10. O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *Proc. of the 8th Ext. Semantic Web Conference (ESWC)*, 2011.
11. O. Hartig. SPARQL for a Web of Linked Data: Semantics and Computability. In *Proc. of the 9th Ext. Semantic Web Conference (ESWC)*, 2012.
12. O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL queries over the web of Linked Data. In *Proc. of the 8th International Semantic Web Conference (ISWC)*, 2009.

13. G. Ladwig and D. T. Tran. Linked Data query processing strategies. In *ISWC*, 2010.
14. G. Ladwig and D. T. Tran. SIHJoin: Querying remote and local linked data. In *ESWC*, 2011.
15. A. O. Mendelzon and T. Milo. Formal models of web queries. *Inf. Systems*, 23(8), 1998.
16. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), 2009.
17. F. Picalausa and S. Vansummeren. What are real SPARQL queries like? In *SWIM*, 2011.
18. M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. In *Proc. of the 13th Int. Conference on Database Theory (ICDT)*, 2010.
19. D. Vrandecić, M. Krötzsch, S. Rudolph, and U. Lösch. Leveraging non-lexical knowledge for the linked open data web. In *RAFT*, 2010.

Appendix

The Appendix is organized as follows:

- Appendix A describes how we encode relevant structures (such as a Web of Linked Data and a valuation) on the tapes of Turing machines.
- Appendix B provides a formal definition of SPARQL.
- Appendix C contains the full technical proofs for all results in the paper.
- Appendix D provides a formal discussion of constant reachability criteria.

A Encoding

To encode Webs of Linked Data and query results on the tapes of a Turing machine we assume the existence of a total order $\prec_{\mathcal{U}}$, $\prec_{\mathcal{B}}$, $\prec_{\mathcal{L}}$, and $\prec_{\mathcal{V}}$ for the URIs in $\mathcal{U}$, the blank nodes in $\mathcal{B}$, the constants in $\mathcal{L}$, and the variables in $\mathcal{V}$, respectively; in all three cases $\prec_x$ could simply be the lexicographic order of corresponding string representations. Furthermore, we assume a total order $\prec_t$ for RDF triples that is based on the aforementioned orders.

For each $u \in \mathcal{U}$, $c \in \mathcal{L}$, and $v \in \mathcal{V}$ let $\mathrm{enc}(u)$, $\mathrm{enc}(c)$, and $\mathrm{enc}(v)$ be the binary representation of u, c, and v, respectively. The encoding of a RDF triple $t = (s, p, o)$, denoted by $\mathrm{enc}(t)$, is a word $\langle\, \mathrm{enc}(s)\,,\, \mathrm{enc}(p)\,,\, \mathrm{enc}(o)\,\rangle$.

The encoding of a finite set of RDF triples $T = \{t_1, \ldots, t_n\}$, denoted by $\mathrm{enc}(T)$, is a word $\langle\!\langle\, \mathrm{enc}(t_1)\,,\, \mathrm{enc}(t_2)\,,\, \ldots\,,\, \mathrm{enc}(t_n)\,\rangle\!\rangle$ where the $\mathrm{enc}(t_i)$ are ordered as follows: For each two RDF triples $t_x, t_y \in T$, $\mathrm{enc}(t_x)$ occurs before $\mathrm{enc}(t_y)$ in $\mathrm{enc}(T)$ if $t_x \prec_t t_y$.

For a Web of Linked Data $W = (D, data, adoc)$, the encoding of LD document $d \in D$, denoted by $\mathrm{enc}(d)$, is the word $\mathrm{enc}(data(d))$. The encoding of W itself, denoted by $\mathrm{enc}(W)$, is a word

$$\sharp\, \mathrm{enc}(u_1)\, \mathrm{enc}(adoc(u_1))\, \sharp \ldots \sharp\, \mathrm{enc}(u_i)\, \mathrm{enc}(adoc(u_i))\, \sharp \ldots$$

where $u_1, \ldots, u_i, \ldots$ is the (potentially infinite but countable) list of URIs in $\mathrm{dom}(adoc)$, ordered according to $\prec_{\mathcal{U}}$.

The encoding of a valuation μ with $dom(\mu) = \{v_1, \ldots, v_n\}$, denoted by $\mathrm{enc}(\mu)$, is a word

$$\langle\!\langle\, \mathrm{enc}(v_1) \to \mathrm{enc}\big(\mu(v_1)\big)\,,\, \ldots\,,\, \mathrm{enc}(v_n) \to \mathrm{enc}\big(\mu(v_n)\big)\,\rangle\!\rangle$$

where the $\mathrm{enc}(\mu(v_i))$ are ordered as follows: For each two variables $v_x, v_y \in dom(\mu)$, $\mathrm{enc}(\mu(v_x))$ occurs before $\mathrm{enc}(\mu(v_y))$ in $\mathrm{enc}(\mu)$ if $v_x \prec_V v_y$.

Finally, the encoding of a (potentially infinite) set of valuations $\Omega = \{\mu_1, \mu_2, \ldots\}$, denoted by $\mathrm{enc}(\Omega)$, is a word $\mathrm{enc}(\mu_1)\, \mathrm{enc}(\mu_2) \ldots$ where the $\mathrm{enc}(\mu_i)$ may occur in any order.

B Formal Definition of SPARQL

A SPARQL *filter condition* is defined recursively as follows: i) If $?x, ?y \in \mathcal{V}$ and $c \in (\mathcal{U} \cup \mathcal{L})$ then $?x = c$, $?x = ?y$, and $\mathrm{bound}(?x)$ are filter conditions; ii) If R_1 and R_2 are filter conditions then $(\neg R_1)$, $(R_1 \wedge R_2)$, and $(R_1 \vee R_2)$ are filter conditions.

Definition 14. *A **SPARQL expression** is defined recursively as follows:*

1. *A tuple $(s, p, o) \in (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L})$ is a SPARQL expression. We call such a tuple a **triple pattern**.*
2. *If P_1 and P_2 are SPARQL expressions, then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, and $(P_1 \text{ OPT } P_2)$ are SPARQL expressions.*
3. *If P' is a SPARQL expression and R is a filter condition, then $(P' \text{ FILTER } R)$ is a SPARQL expression.*

Let μ be a valuation and let R be a filter condition. We say μ satisfies R iff either i) R is $?x = c$, $?x \in \text{dom}(\mu)$ and $\mu(?x) = c$; ii) R is $?x = ?y$, $?x, ?y \in \text{dom}(\mu)$ and $\mu(?x) = \mu(?y)$; iii) R is $\text{bound}(?x)$ and $?x \in \text{dom}(\mu)$; iv) R is $(\neg R_1)$ and μ does not satisfy R_1; v) R is $(R_1 \wedge R_2)$ and μ satisfies R_1 and R_2; or vi) R is $(R_1 \vee R_2)$ and μ satisfies R_1 or R_2.

Let Ω_l, Ω_r and Ω be (potentially infinite but countable) sets of valuations; let R be a filter condition. The binary operations *join*, *union*, *difference*, and *left outer-join* between Ω_l and Ω_r are defined as follows:

$$\begin{aligned}
\Omega_l \bowtie \Omega_r &= \{\mu_l \cup \mu_r \mid \mu_l \in \Omega_l \text{ and } \mu_r \in \Omega_r \text{ and } \mu_l \sim \mu_r\} \\
\Omega_l \cup \Omega_r &= \{\mu \mid \mu \in \Omega_l \text{ or } \mu \in \Omega_r\} \\
\Omega_l \setminus \Omega_r &= \{\mu_l \in \Omega_l \mid \forall \mu_r \in \Omega_r : \mu_l \not\sim \mu_r\} \\
\Omega_l \mathbin{⟕} \Omega_r &= (\Omega_l \bowtie \Omega_r) \cup (\Omega_l \setminus \Omega_r) \\
\sigma_R(\Omega) &= \{\mu \in \Omega \mid \mu \text{ satisfies } R\}
\end{aligned}$$

Definition 15. *Let P be a SPARQL expression and let G be a (potentially infinite but countable) set of RDF triples. The **evaluation of P over** G, denoted by $[\![P]\!]_G$, is defined recursively as follows:*

1. *If P is a triple pattern tp, then*

$$[\![P]\!]_G = \{\mu \mid \mu \text{ is a valuation with } \text{dom}(\mu) = \text{vars}(tp) \text{ and } \mu[tp] \in G\}$$

2. *If P is $(P_1 \text{ AND } P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$.*
3. *If P is $(P_1 \text{ UNION } P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$.*
4. *If P is $(P_1 \text{ OPT } P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$.*
5. *If P is $(P' \text{ FILTER } R)$, then $[\![P]\!]_G = \sigma_R([\![P']\!]_G)$.*

*Each valuation $\mu \in [\![P]\!]_G$ is called a **solution** for P in G.*

C Proofs

C.1 Additional References for the Proofs

[Pap93] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.

C.2 Proof of Proposition 1, Case 1

For this proof we introduce a notion of lineage for valuations. Informally, the lineage of a valuation μ is the set of all RDF triples that are required to construct μ. Formally:

Definition 16. *Let P be a SPARQL expression and G be a (potentially infinite) set of RDF triples such that $[\![P]\!]_G \neq \varnothing$. For each $\mu \in [\![P]\!]_G$ we define the (P, G)-**lineage of** μ, denoted by $\mathrm{lin}^{P,G}(\mu)$, recursively as follows:*

1. *If P is a triple pattern tp, then $\mathrm{lin}^{P,G}(\mu) = \{\mu[tp]\}$.*
2. *If P is $(P_1 \text{ AND } P_2)$, then*

$$\mathrm{lin}^{P,G}(\mu) = \mathrm{lin}^{P_1,G}(\mu_1) \cup \mathrm{lin}^{P_2,G}(\mu_2)$$

   where $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$ such that $\mu_1 \sim \mu_2$ and $\mu = \mu_1 \cup \mu_2$. Notice, μ_1 and μ_2 must exist because $\mu \in [\![P]\!]_G$.
3. *If P is $(P_1 \text{ UNION } P_2)$, then*

$$\mathrm{lin}^{P,G}(\mu) = \begin{cases} \mathrm{lin}^{P_1,G}(\mu_1) & \text{if } \exists\, \mu_1 \in [\![P_1]\!]_G : \mu_1 = \mu, \\ \mathrm{lin}^{P_2,G}(\mu_2) & \text{if } \exists\, \mu_2 \in [\![P_2]\!]_G : \mu_2 = \mu. \end{cases}$$

   Notice, if μ_1 does not exist then μ_2 must exist because $\mu \in [\![P]\!]_G$.
4. *If P is $(P_1 \text{ OPT } P_2)$, then*

$$\mathrm{lin}^{P,G}(\mu) = \begin{cases} \mathrm{lin}^{P_1,G}(\mu_1) \cup \mathrm{lin}^{P_2,G}(\mu_2) & \text{if } \exists\, (\mu_1, \mu_2) \in [\![P_1]\!]_G \times [\![P_2]\!]_G : \big(\mu_1 \sim \mu_2 \wedge \mu = \mu_1 \cup \mu_2\big), \\ \mathrm{lin}^{P_1,G}(\mu') & \text{if } \exists\, \mu' \in [\![P_1]\!]_G : \big(\mu' = \mu \wedge \forall\, \mu^* \in [\![P_2]\!]_G : \mu^* \not\sim \mu'\big). \end{cases}$$

   Notice, either μ_1 and μ_2 or μ' must exist because $\mu \in [\![P]\!]_G$.
5. *If P is $(P' \text{ FILTER } R)$, then $\mathrm{lin}^{P,G}(\mu) = \mathrm{lin}^{P',G}(\mu')$ where $\mu' \in [\![P']\!]_G$ such that $\mu = \mu'$. Notice, μ' must exist because $\mu \in [\![P]\!]_G$.*

For any SPARQL expression P, any (potentially infinite) set G of RDF triples, and any valuation $\mu \in [\![P]\!]_G$ it can be easily seen that i) $G' = \mathrm{lin}^{P,G}(\mu)$ is finite and ii) $\mu \in [\![P]\!]_{G'}$. We now prove Proposition 1, case 1:

If: Let P be a SPARQL expression that is satisfiable. Hence, there exists a set of RDF triples G such that $[\![P]\!]_G \neq \varnothing$. W.l.o.g., let μ be an arbitrary solution for P in G, that is, $\mu \in [\![P]\!]_G$. Furthermore, let $G' = \mathrm{lin}^{P,G}(\mu)$ be the (P, G)-lineage of μ. We use G' to construct a Web of Linked Data $W_\mu = (D_\mu, data_\mu, adoc_\mu)$ which consists of a single LD document. This document may be retrieved using any URI and it contains the (P, G)-lineage of μ (recall that the lineage is guaranteed to be a finite). Formally:

$$D_\mu = \{d\} \qquad data_\mu(d) = G' \qquad \forall\, u \in \mathcal{U} : adoc_\mu(u) = d$$

We now consider the result of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ (which uses P) over W_μ. Obviously, $\mathrm{AllData}(W_\mu) = G'$ and, thus, $\mathcal{Q}^P\big(W_\mu\big) = [\![P]\!]_{G'}$ (cf. Definition 8). Since we know $\mu \in [\![P]\!]_{G'}$ it holds $\mathcal{Q}^P\big(W_\mu\big) \neq \varnothing$, which shows that $\mathcal{Q}^P$ is satisfiable.

Only if: Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P. Since $\mathcal{Q}^P$ is satisfiable, exists a Web of Linked Data W such that $\mathcal{Q}^P\big(W\big) \neq \varnothing$. Since $\mathcal{Q}^P\big(W\big) = [\![P]\!]_{\mathrm{AllData}(W)}$ (cf. Definition 8), we conclude that P is satisfiable.

C.3 Proof of Proposition 1, Case 2

We prove case 2 of Proposition 1 using the same argumentation that we use in Section C.2 for case 1.

If: Let P be a SPARQL expression that is nontrivially satisfiable. Hence, there exists a set of RDF triples G and a valuation μ such that i) $\mu \in [\![P]\!]_G$ and ii) $\mathrm{dom}(\mu) \neq \varnothing$. Let $G' = \mathrm{lin}^{P,G}(\mu)$ be the (P,G)-lineage of μ. We use G' to construct a Web of Linked Data $W_\mu = (D_\mu, data_\mu, adoc_\mu)$ which consists of a single LD document. This document may be retrieved using any URI and it contains the (P,G)-lineage of μ (recall that the lineage is guaranteed to be a finite). Formally:

$$D_\mu = \{d\} \qquad data_\mu(d) = G' \qquad \forall\, u \in \mathcal{U} : adoc_\mu(u) = d$$

We now consider the result of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ (which uses P) over W_μ. Obviously, $\mathrm{AllData}(W_\mu) = G'$ and, thus, $\mathcal{Q}^P\big(W_\mu\big) = [\![P]\!]_{G'}$ (cf. Definition 8). Since we know $\mu \in [\![P]\!]_{G'}$ and $\mathrm{dom}(\mu) \neq \varnothing$, we conclude that $\mathcal{Q}^P$ is nontrivially satisfiable.

Only if: Let $\mathcal{Q}^P$ be a nontrivially satisfiable SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P. Since $\mathcal{Q}^P$ is nontrivially satisfiable, exists a Web of Linked Data W and a valuation μ such that i) $\mu \in \mathcal{Q}^P\big(W\big)$ and ii) $\mathrm{dom}(\mu) \neq \varnothing$. Since $\mathcal{Q}^P\big(W\big) = [\![P]\!]_{\mathrm{AllData}(W)}$ (cf. Definition 8), we conclude that P is nontrivially satisfiable.

C.4 Proof of Proposition 1, Case 3

If: Let:

- P be a SPARQL expression that is monotonic;
- $\mathcal{Q}^P$ be the SPARQL$_{\mathsf{LD}}$ query that uses P; and
- W_1, W_2 be an arbitrary pair of Webs of Linked Data such that W_1 is an induced subweb of W_2.

To prove that $\mathcal{Q}^P$ is monotonic it suffices to show $\mathcal{Q}^P\big(W_1\big) \subseteq \mathcal{Q}^P\big(W_2\big)$. According to Definition 8 we have $\mathcal{Q}^P\big(W_1\big) = [\![P]\!]_{\mathrm{AllData}(W_1)}$ and $\mathcal{Q}^P\big(W_2\big) = [\![P]\!]_{\mathrm{AllData}(W_2)}$. Since W_1 is an induced subweb of W_2 it holds $\mathrm{AllData}(W_1) \subseteq \mathrm{AllData}(W_2)$. We may now use the monotonicity of P to show $[\![P]\!]_{\mathrm{AllData}(W_1)} \subseteq [\![P]\!]_{\mathrm{AllData}(W_2)}$. Hence, $\mathcal{Q}^P\big(W_1\big) \subseteq \mathcal{Q}^P\big(W_2\big)$.

Only if: Let:

- $\mathcal{Q}^P$ be a monotonic SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P; and
- G_1, G_2 be an arbitrary pair of set of RDF triples such that $G_1 \subseteq G_2$.

We distinguish two cases: either P is satisfiable or P is not satisfiable. In the latter case P is trivially monotonic. Hence, we only have to discuss the first case. To prove that (the satisfiable) P is monotonic it suffices to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$.

Similar to the proof for the other direction, we aim to use G_1 and G_2 for constructing two Webs of Linked Data W_1 and W_2 (where W_1 is an induced subweb of W_2) and then use the monotonicity of $\mathcal{Q}^P$ to show the monotonicity of P. However, since G_1 and

G_2 may be (countably) infinite we cannot simply construct Webs of Linked Data that consist of single LD documents which contain all RDF triples of G_1 and G_2 (recall, the data in each LD document of a Web of Linked Data must be finite). As an alternative strategy we construct Webs of Linked Data that consists of as many LD documents as we have RDF triples in G_1 and G_2 (which may be infinitely many). However, since the data of each LD document in a Web of Linked Data must use a unique set of blank nodes, we may lose certain solutions $\mu \in [\![P]\!]_{G_1}$ by distributing the RDF triples from G_1 over multiple LD documents; similarly for G_2. To avoid this issue we assume i) a set $U_B \subset \mathcal{U}$ of new URIs not mentioned in G_2 (i.e. $U_B \cap \mathrm{terms}(G_2) = \varnothing$) and ii) a bijective mapping $\varrho : \mathrm{terms}(G_2) \to \big(U_B \cup \mathrm{terms}(G_2) \cap (\mathcal{U} \cup \mathcal{L})\big)$ that, for any $x \in \mathrm{terms}(G_2)$, is defined as follows:

$$\varrho(x) = \begin{cases} \varrho_B(x) & \text{if } x \in (\mathrm{terms}(G_2) \cap \mathcal{B}), \\ x & \text{else.} \end{cases}$$

where $\varrho_B : (\mathrm{terms}(G_2) \cap \mathcal{B}) \to U_B$ is an arbitrary bijection that maps each blank node in G_2 to a new, unique URI $u \in U_B$.

The application of ϱ to an arbitrary valuation μ, denoted by $\varrho[\mu]$, results in a valuation μ' such that $\mathrm{dom}(\mu') = \mathrm{dom}(\mu)$ and $\mu'(?v) = \varrho(\mu(?v))$ for all $?v \in \mathrm{dom}(\mu)$. Furthermore, the application of ϱ to an arbitrary RDF triple $t = (x_1, x_2, x_3)$, denoted by $\varrho[t]$, results in an RDF triple $t' = (x'_1, x'_2, x'_3)$ such that $x'_i = \varrho(x_i)$ for all $i \in \{1, 2, 3\}$. We now let $G'_1 = \{\varrho[t] \,|\, t \in G_1\}$ and $G'_2 = \{\varrho[t] \,|\, t \in G_2\}$. The following facts are verified easily:

Fact 2. *It holds $G'_1 \subseteq G'_2$, $|G_1| = |G'_1|$, and $|G_2| = |G'_2|$.*

Fact 3. *For all $j \in \{1, 2\}$ it holds: Let μ be an arbitrary valuation, then $\mu' = \varrho[\mu]$ is a solution for P in G'_j if and only if μ is a solution for P in G_j. More precisely:*

$$\forall\, \mu \in [\![P]\!]_{G_j} : \varrho[\mu] \in [\![P]\!]_{G'_j} \qquad \textit{and} \qquad \forall\, \mu' \in [\![P]\!]_{G'_j} : \varrho^{-1}[\mu'] \in [\![P]\!]_{G_j}$$

where ϱ^{-1} denotes the inverse of the bijective mapping ϱ.

We now use G'_2 to construct a Web of Linked Data $W_2 = (D_2, data_2, adoc_2)$ as follows: D_2 consists of $|G'_2|$ LD documents, each of which contains a particular RDF triple from G'_2. Furthermore, we assume a set U_2 of URIs, each of which corresponds to a particular RDF triple from G'_2; hence, $U_2 \subset \mathcal{U}$ and $|U_2| = |G'_2|$. These URIs may be used to retrieve the LD document for the corresponding RDF triple. For a formal definition let d_{t_i} denote the LD document for RDF triple $t_i \in G'_2$ and let u_{t_i} denote the URI that corresponds to $t_i \in G'_2$. Then, we let:

$$D_2 = \bigcup_{t_i \in G'_2} d_{t_i} \qquad data_2(d_{t_i}) = \{t_i\} \qquad \forall\, u_{t_i} \in U_2 : adoc_2(u_{t_i}) = d_{t_i}$$

In addition to W_2, we introduce a Web of Linked Data $W_1 = (D_1, data_1, adoc_1)$ that is an induced subweb of W_2 and that is defined by $D_1 = \{d_{t_i} \in D_2 \,|\, t_i \in G'_1\}$. Recall, any induced subweb is unambiguously defined by specifying its set of LD documents. It can be easily seen that $\mathrm{AllData}(W_1) = G'_1$ and $\mathrm{AllData}(W_2) = G'_2$.

We now use W_1 and W_2 and the monotonicity of $\mathcal{Q}^P$ to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ (which proves that P is monotonic). W.l.o.g., let μ be an arbitrary solution for P in G_1, that is, $\mu \in [\![P]\!]_{G_1}$. Notice, such a μ must exist because we assume that P is satisfiable (see before). To prove $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ it suffices to show $\mu \in [\![P]\!]_{G_2}$.

Due to Fact 3 it holds $\varrho[\mu] \in [\![P]\!]_{G'_1}$; and with $\mathrm{AllData}(W_1) = G'_1$ and Definition 8 we have $[\![P]\!]_{G'_1} = [\![P]\!]_{\mathrm{AllData}(W_1)} = \mathcal{Q}^P\big(W_1\big)$. Since W_1 is an induced subweb of W_2 and $\mathcal{Q}^P$ is monotonic, it holds $\mathcal{Q}^P\big(W_1\big) \subseteq \mathcal{Q}^P\big(W_2\big)$. We now use $\mathrm{AllData}(W_2) = G'_2$ to show $\varrho[\mu] \in [\![P]\!]_{G'_2}$. Finally, we use Fact 3 again and find $\varrho^{-1}\big[\varrho[\mu]\big] = \mu \in [\![P]\!]_{G_2}$.

C.5 Proof of Theorem 1

To prove the theorem we first show that not any satisfiable SPARQL$_{\mathsf{LD}}$ query is finitely computable. Next, we study SPARQL$_{\mathsf{LD}}$ queries that are not monotonic and show that these queries are not eventually computable (and, thus, not computable at all). Finally, we prove that satisfiable, monotonic SPARQL$_{\mathsf{LD}}$ queries are eventually computable.

To show that not any satisfiable SPARQL$_{\mathsf{LD}}$ query is finitely computable we use a contradiction, that is, we assume a satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ which is finitely computable and show that this assumption must be false. If $\mathcal{Q}^P$ were finitely computable, there would be an LD machine that, for any Web of Linked Data W encoded on the Web tape, halts after a finite number of computation steps and produces a possible encoding of $\mathcal{Q}^P\big(W\big)$ on its output tape (cf. Definition 5). To obtain a contradiction we show that such a machine does not exist. However, for the proof we assume M were such a machine.

To compute $\mathcal{Q}^P$ over an arbitrary Web of Linked Data $W = (D, data, adoc)$ (which is encoded on the Web tape of M), machine M requires access to the data of all LD documents $d \in D$ (Recall that $\mathcal{Q}^P\big(W\big) = [\![P]\!]_{\mathrm{AllData}(W)}$ where $\mathrm{AllData}(W) = \big\{data(d) \,|\, d \in D\big\}$). However, M may only access an LD document $d \in D$ (and its data) after entering the expand state with a corresponding URI $u \in \mathcal{U}$ on the link traversal tape (i.e. for u it must hold $adoc(u) = d$). Initially, the machine has no information about which URI(s) to use for accessing any $d \in D$. Hence, to ensure that all $d \in D$ have been accessed, M must expand all $u \in \mathcal{U}$. Notice, a real query system for the WWW would have to perform a similar procedure: To guarantee that such a system sees all documents, it must enumerate and lookup all URIs. However, since $\mathcal{U}$ is (countably) infinite, this process does not terminate, which is a contradiction to our assumption that M halts after a finite number of computation steps. Hence, satisfiable SPARQL$_{\mathsf{LD}}$ queries cannot be finitely computable.

We now show by contradiction that non-monotonic SPARQL$_{\mathsf{LD}}$ queries are not eventually computable. To obtain a contradiction we assume a (satisfiable) SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ that is not monotonic and an LD machine M whose computation of $\mathcal{Q}^P$ on any Web of Linked Data W has the two properties given in Definition 6. Let $W = (D, data, adoc)$ be a Web of Linked Data such that $\mathcal{Q}^P\big(W\big) \neq \varnothing$; such a Web exists because $\mathcal{Q}^P$ is satisfiable. Let W be encoded on the Web tape of M and let μ be an arbitrary solution for $\mathcal{Q}^P$ in W; i.e. $\mu \in \mathcal{Q}^P\big(W\big)$. Based on our assumption, machine M must write $\mathrm{enc}(\mu)$ to its output tape after a finite number of computation steps (cf. property 2 in Definition 6). We argue that this is impossible: Since $\mathcal{Q}^P$ is not monotonic, M

Algorithm 1 The program of the (P)-machine.

1: $j := 1$
2: **for** $u \in \mathcal{U}$ **do**
3: Call `lookup` for u.
4: Let T_j denote the set of all RDF triples currently encoded on the link traversal tape. Use the work tape to enumerate the set $[\![P]\!]_{T_j}$.
5: For each $\mu \in [\![P]\!]_{T_j}$ check whether μ is already encoded on the output tape; if not, then add $\text{enc}(\mu)$ to the output.
6: $j := j + 1$
7: **end for**

cannot add μ to the output before it is guaranteed that all $d \in D$ have been accessed. As discussed before, such a guarantee requires expanding all $u \in \mathcal{U}$ because M has no a-priory information about W. However, expanding all $u \in \mathcal{U}$ is a non-terminating process (due to the infiniteness of $\mathcal{U}$) and, thus, M does not write μ to its output after a finite number of steps. As a consequence, the computation of $\mathcal{Q}^P(W)$ by M does not have the properties given in Definition 6, which contradicts our initial assumption. This contradiction shows that non-monotonic SPARQL$_{\mathsf{LD}}$ queries are not eventually computable.

In the remainder, we prove that satisfiable, monotonic SPARQL$_{\mathsf{LD}}$ queries are eventually computable. For this proof we introduce specific LD machines which we call (P)-machines. Such a (P)-machine implements a generic (i.e. input independent) computation of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$. We shall see that if a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic, the corresponding (P)-machine (eventually) computes $\mathcal{Q}^P$ over any Web of Linked Data. Formally, we define (P)-machines as follows:

Definition 17. *Let P be SPARQL expression. The (P)-**machine** is an LD machine that implements Algorithm 1. This algorithm makes use of a special subroutine called `lookup`, which, when called with URI $u \in \mathcal{U}$, i) writes $\text{enc}(u)$ to the right end of the word on the link traversal tape, ii) enters the expand state, and iii) performs the expand procedure as specified in Definition 4.*

Before we complete the proof we discuss important properties of (P)-machines. As can be seen in Algorithm 1, any computation performed by (P)-machines enters a loop that iterates over the set $\mathcal{U}$ of all possible URIs. As discussed before, expanding all $u \in \mathcal{U}$ is necessary to guarantee completeness of the computed query result. However, since $\mathcal{U}$ is (countably) infinite the algorithm does not terminate (which is not a requirement for eventual computability).

During each iteration of its main processing loop, a (P)-machine generates valuations using all data that is currently encoded on the link traversal tape. The following lemma shows that these valuations are part of the corresponding query result (find the proof for Lemma 3 below in Section C.6):

Lemma 3. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the (P)-machine for SPARQL expression P used by $\mathcal{Q}^P$; and let W be an arbitrary Web of Linked Data encoded on the Web tape of M^P. During the execution of Algorithm 1 by M^P it holds $\forall\, j \in \{1, 2, ...\} : [\![P]\!]_{T_j} \subseteq \mathcal{Q}^P(W)$.*

We now use the notion of (P)-machines to prove that satisfiable, monotonic SPARQL$_{\mathsf{LD}}$ queries are eventually computable. Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic and let W be an arbitrary Web of Linked Data encoded on the Web tape of the (P)-machine for $\mathcal{Q}^P$; to denote this machine we write M^P. W.l.o.g. it suffices to show that the computation of M^P on (Web) input $\mathrm{enc}(W)$ has the two properties given in Definition 6.

During the computation M^P only writes to its output tape when it adds (encoded) valuations $\mu \in [\![P]\!]_{T_j}$ (for $j = 1, 2, ...$). Since all these valuations are solutions for $\mathcal{Q}^P$ in W (cf. Lemma 3) and line 5 in Algorithm 1 ensures that the output is free of duplicates, we see that the word on the output tape is always a prefix of a possible encoding of $\mathcal{Q}^P\big(W\big)$. Hence, the computation of M^P has the first property specified in Definition 6.

To verify that the computation also has the second property it is important to note that Algorithm 1 looks up no more than one URI per iteration (cf. line 3). Hence, (P)-machines prioritize result construction over data retrieval. This feature allows us to show that for each solution in a query result exists an iteration during which that solution is computed (find the proof for Lemma 4 below in Section C.7):

Lemma 4. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the (P)-machine for SPARQL expression P used by $\mathcal{Q}^P$; and let W be an arbitrary Web of Linked Data encoded on the Web tape of M^P. For each $\mu \in \mathcal{Q}^P\big(W\big)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 1 by M^P it holds $\forall\, j \in \{j_\mu, j_\mu+1, ...\} : \mu \in [\![P]\!]_{T_j}$.*

It remains to show that the computation of M^P definitely reaches each iteration of the processing loop after a finite number of computation steps. To prove this property we show that each iteration of the loop finishes after a finite number of computation steps:

- The call of the subroutine `lookup` (cf. Definition 17) in line 3 of Algorithm 1 terminates because the encoding of $W = (D, data, adoc)$ is ordered following the order of the URIs in $\mathrm{dom}(adoc)$.
- At any point in the computation the word on the link traversal tape is finite because M^P only gradually appends (encoded) LD documents to the link traversal tape and the encoding of each document is finite (recall that the set of RDF triples $data(d)$ for each LD document d is finite). Due to the finiteness of the word on the link traversal tape, each $[\![P]\!]_{T_j}$ (for $j = 1, 2, ...$) is finite, resulting in a finite number of computation steps for line 4 during any iteration.
- Finally, line 5 requires only a finite number of computation steps because the word on the link traversal tape is finite at any point in the computation; so is the word on the output tape.

C.6 Proof of Lemma 3

Let:

- $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that is satisfiable and monotonic;
- M^P denote the (P)-machine for SPARQL expression P used by $\mathcal{Q}^P$;

– W be a Web of Linked Data which is encoded on the Web tape of M^P.

To prove Lemma 3 we use the following result.

Lemma 5. *During the execution of Algorithm 1 by M^P on (Web) input $\mathrm{enc}(W)$ it holds $\forall\, j \in \{1, 2, ...\} : T_j \subseteq \mathrm{AllData}(W)$.*

Proof of Lemma 5. The computation of M^P starts with an empty link traversal tape (cf. Definition 4). Let w_j be the word on the link traversal tape of M^P before M^P executes line 4 during the j-th iteration of the main processing loop in Algorithm 1. It can be easily seen that for each w_j (where $j \in \{1, 2, ...\}$) exists a finite sequence $u_1, ..., u_j$ of j different URIs such that i) w_j is[6]

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,...\,\sharp\,\mathrm{enc}(u_j)\,\mathrm{enc}(adoc(u_j))\,\sharp$$

and ii) for each $i \in [1, j]$ either $u_i \notin \mathrm{dom}(adoc)$ or $adoc(u_i) \in D$. If U_j is the set that contains all URIs in this sequence $u_1, ..., u_j$, it holds $T_j = \big\{data\,(adoc(u_i)) \;\big|\; u_i \in U_j \text{ and } u_i \in \mathrm{dom}(adoc)\big\}$. Clearly, $T_j \subseteq \mathrm{AllData}(W)$. □

Due to the monotonicity of $\mathcal{Q}^P$ it is trivial to show Lemma 3 using Lemma 5 (recall $\mathcal{Q}^P(W) = [\![P]\!]_{\mathrm{AllData}(W)}$).

C.7 Proof of Lemma 4

Let:

– $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that is satisfiable and monotonic;
– M^P denote the (P)-machine for SPARQL expression P used by $\mathcal{Q}^P$;
– W be a Web of Linked Data which is encoded on the Web tape of M^P.

To prove Lemma 4 we use the following result.

Lemma 6. *For each RDF triple $t \in \mathrm{AllData}(W)$ exists a $j_t \in \{1, 2, ...\}$ such that during the execution of Algorithm 1 by M^P on (Web) input $\mathrm{enc}(W)$ it holds $\forall\, j \in \{j_t, j_t{+}1, ...\} : t \in T_j$.*

Proof of Lemma 6. W.l.o.g., let t' be an arbitrary RDF triple $t' \in \mathrm{AllData}(W)$; hence, there exists an LD document $d \in D$ such that $t' \in data(d)$. Let d' be such an LD document. Since mapping $adoc$ is surjective (cf. Definition 1), exists a URI $u \in \mathcal{U}$ such that $adoc(u) = d'$. Let u' be such a URI. Since $u' \in \mathcal{U}$ exists a $j' \in \{1, 2, ...\}$ such that M^P selects u' for processing in the j'-th iteration of the main loop in Algorithm 1. After completing the lookup of u' during this iteration (cf. line 3 in Algorithm 1), the word on the link traversal tape contains sub-word $\mathrm{enc}(d')$ (cf. Definitions 17 and 4). Since $t' \in data(d')$, this word $\mathrm{enc}(d')$ contains sub-word $\mathrm{enc}(t')$ (cf. Appendix A). Hence, $t' \in T_{j'}$. Since (P)-machines only append to (the right end of) the word on the link traversal tape, M^P will never remove $\mathrm{enc}(t')$ from that tape and, thus, it holds $\forall\, j \in \{j', j'{+}1, ...\} : t' \in T_j$. □

[6] We assume $\mathrm{enc}(adoc(u_i))$ is the empty word if $u_i \notin \mathrm{dom}(adoc)$.

We now prove Lemma 4 by induction over the structure of possible SPARQL expressions.

Base case: Assume that SPARQL expression P is a triple pattern tp. W.l.o.g., let $\mu \in \mathcal{Q}^P(W)$. It holds $\mathrm{dom}(\mu) = \mathrm{vars}(tp)$ and $t = \mu[tp] \in \mathrm{AllData}(W)$ (cf. Definitions 8 and 15). According to Lemma 6 exists a $j_\mu \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_\mu, j_\mu{+}1, ...\}$: $t \in T_j$. Since $\mathcal{Q}^P$ is monotonic we conclude $\forall\, j \in \{j_\mu, j_\mu{+}1, ...\} : \mu \in [\![P]\!]_{T_j}$.

Induction step: Our inductive hypothesis is that for SPARQL expressions P_1 and P_2 it holds:

1. For each $\mu \in \mathcal{Q}^{P_1}(W)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 1 by M^P it holds $\forall\, j \in \{j_\mu, j_\mu{+}1, ...\} : \mu \in [\![P_1]\!]_{T_j}$; and
2. For each $\mu \in \mathcal{Q}^{P_2}(W)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 1 by M^P it holds $\forall\, j \in \{j_\mu, j_\mu{+}1, ...\} : \mu \in [\![P_2]\!]_{T_j}$.

Based on this hypothesis we show that for any SPARQL expression P that can be constructed using P_1 and P_2 it holds: For each $\mu \in \mathcal{Q}^P(W)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 1 by M^P it holds $\forall\, j \in \{j_\mu, j_\mu{+}1, ...\}$: $\mu \in [\![P]\!]_{T_j}$. W.l.o.g., let $\mu' \in \mathcal{Q}^P(W)$. According to Definition 14 we distinguish the following cases:

- P is $(P_1 \text{ AND } P_2)$. In this case exist $\mu_1 \in \mathcal{Q}^{P_1}(W)$ and $\mu_2 \in \mathcal{Q}^{P_2}(W)$ such that $\mu' = \mu_1 \cup \mu_2$ and $\mu_1 \sim \mu_2$. According to our inductive hypothesis exist $j_{\mu_1}, j_{\mu_2} \in \{1, 2, ...\}$ such that i) $\forall\, j \in \{j_{\mu_1}, j_{\mu_1}{+}1, ...\} : \mu_1 \in [\![P_1]\!]_{T_j}$ and ii) $\forall\, j \in \{j_{\mu_2}, j_{\mu_2}{+}1, ...\} : \mu_2 \in [\![P_2]\!]_{T_j}$. Let $j_{\mu'} = \max(\{j_{\mu_1}, j_{\mu_2}\})$. Due to the monotonicity of $\mathcal{Q}^P$ it holds $\forall\, j \in \{j_{\mu'}, j_{\mu'}{+}1, ...\} : \mu' \in [\![P]\!]_{T_j}$.
- P is $(P_1 \text{ FILTER } R)$. In this case exist $\mu^* \in \mathcal{Q}^{P_1}(W)$ such that $\mu' = \mu^*$. According to our inductive hypothesis exist $j_{\mu^*} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}{+}1, ...\} : \mu^* \in [\![P_1]\!]_{T_j}$. Due to the monotonicity of $\mathcal{Q}^P$ it holds $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}{+}1, ...\} : \mu' \in [\![P]\!]_{T_j}$.
- P is $(P_1 \text{ OPT } P_2)$. We distinguish two cases:
  1. There exist $\mu_1 \in \mathcal{Q}^{P_1}(W)$ and $\mu_2 \in \mathcal{Q}^{P_2}(W)$ such that $\mu' = \mu_1 \cup \mu_2$ and $\mu_1 \sim \mu_2$. This case corresponds to the case where P is $(P_1 \text{ AND } P_2)$ (see above).
  2. There exist $\mu_1 \in \mathcal{Q}^{P_1}(W)$ such that $\mu' = \mu_1$ and $\forall\, \mu_2 \in \mathcal{Q}^{P_2}(W) : \mu_1 \not\sim \mu_2$. According to our inductive hypothesis exist $j_{\mu_1} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu_1}, j_{\mu_1} + 1, ...\} : \mu_1 \in [\![P_1]\!]_{T_j}$. Due to the monotonicity of $\mathcal{Q}^P$ it holds $\forall\, j \in \{j_{\mu_1}, j_{\mu_1}{+}1, ...\} : \mu' \in [\![P]\!]_{T_j}$.
- P is $(P_1 \text{ UNION } P_2)$. We distinguish two cases:
  1. There exists $\mu^* \in \mathcal{Q}^{P_1}(W)$ such that $\mu' = \mu^*$. According to our inductive hypothesis exist $j_{\mu^*} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}{+}1, ...\} : \mu^* \in [\![P_1]\!]_{T_j}$.
  2. There exists $\mu^* \in \mathcal{Q}^{P_2}(W)$ such that $\mu' = \mu^*$. According to our inductive hypothesis exist $j_{\mu^*} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}{+}1, ...\} : \mu^* \in [\![P_2]\!]_{T_j}$.

  Due to the monotonicity of $\mathcal{Q}^P$ it holds for both cases: $\forall\, j \in \{j_{\mu^*}, j_{\mu^*} + 1, ...\}$: $\mu' \in [\![P]\!]_{T_j}$.

C.8 Proof of Lemma 1

We prove the lemma by contradiction, that is, we assume a nontrivially satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ for which exists an LD machine that, for some Web of Linked Data W encoded on the Web tape, halts after a finite number of computation steps and produces an encoding of $\mathcal{Q}^P(W)$ on its output tape. To obtain a contradiction we show that such an LD machine and such a Web of Linked Data does not exist. However, for the proof we assume M' were such a machine and $W' = (D', data', adoc')$ were such a Web of Linked Data.

Since $\mathcal{Q}^P$ is nontrivially satisfiable it is possible that W' is a Web of Linked Data for which exist solutions in $\mathcal{Q}^P(W')$ such that each of these solutions provides a binding for at least one variable. Hence, for computing $\mathcal{Q}^P$ over W' completely, machine M' requires access to the data of all LD documents $d \in D'$ (recall that $\mathcal{Q}^P(W') = [\![P]\!]_{\mathrm{AllData}(W')}$ where $\mathrm{AllData}(W') = \{data'(d) \mid d \in D'\}$). However, M' may access an LD document $d \in D'$ (and its data) only after performing the expand procedure with a corresponding URI $u \in \mathcal{U}$ on the link traversal tape (i.e. for u it must hold $adoc'(u) = d$). Initially, the machine has no information about which URI(s) to use for accessing any $d \in D'$. Hence, to ensure that all $d \in D'$ have been accessed, M' must expand all $u \in \mathcal{U}$. Notice, a real query system for the WWW would have to perform a similar procedure: To guarantee that such a system sees all documents, it must enumerate and lookup all URIs. However, since $\mathcal{U}$ is (countably) infinite, this process does not terminate, which is a contradiction to our assumption that M' halts after a finite number of computation steps.

C.9 Proof of Theorem 2

We formally define the termination problem for SPARQL$_{\mathsf{LD}}$ as follows:

Problem:	TERMINATION(SPARQL$_{\mathsf{LD}}$)
Web Input:	a Web of Linked Data W
Ordinary Input:	a satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Does an LD machine exist that computes $\mathcal{Q}^P(W)$ and halts?

We show that TERMINATION(SPARQL$_{\mathsf{LD}}$) is not LD machine decidable by reducing the halting problem to TERMINATION(SPARQL$_{\mathsf{LD}}$).

The halting problem asks whether a given Turing machine (TM) halts on a given input. For the reduction we assume an infinite Web of Linked Data W_{TMs} which we define in the following. Informally, W_{TMs} describes all possible computations of all TMs. For a formal definition of W_{TMs} we adopt the usual approach to unambiguously describe TMs and their input by finite words over the (finite) alphabet of a universal TM (e.g. [Pap93]). Let $\mathcal{W}$ be the countably infinite set of all words that describe TMs. For each $w \in \mathcal{W}$ let $M(w)$ denote the machine described by w, let $c^{w,x}$ denote the computation of $M(w)$ on input x, and let $u^{w,x}$ denote a URI that identifies $c^{w,x}$. Furthermore, let $u_i^{w,x}$ denote a URI that identifies the i-th step in computation $c^{w,x}$. To denote the (infinite) set of all such URIs we write $\mathcal{U}_{\mathsf{TMsteps}}$. Using the URIs $\mathcal{U}_{\mathsf{TMsteps}}$ we may unambiguously identify each step in each possible computation of any TM on

any given input. However, if a URI $u \in \mathcal{U}$ could potentially identify a computation step of a TM on some input (because u adheres to the pattern used for such URIs) but the corresponding step may never exist, then $u \notin \mathcal{U}_{\mathsf{TMsteps}}$. For instance, if the computation of a particular TM $M(w_j)$ on a particular input x_k halts with the i'-th step, then $\forall\, i \in \{1, \ldots, i'\} : u_i^{w_j,x_k} \in \mathcal{U}_{\mathsf{TMsteps}}$ and $\forall\, i \in \{i'+1, \ldots\} : u_i^{w_j,x_k} \notin \mathcal{U}_{\mathsf{TMsteps}}$. Notice, while the set $\mathcal{U}_{\mathsf{TMsteps}}$ is infinite, it is still countable because i) $\mathcal{W}$ is countably infinite, ii) the set of all possible input words for TMs is countably infinite, and iii) i is a natural number.

We now define W_{TMs} as a Web of Linked Data $(D_{\mathsf{TMs}}, data_{\mathsf{TMs}}, adoc_{\mathsf{TMs}})$ with the following elements: D_{TMs} consists of $|\mathcal{U}_{\mathsf{TMsteps}}|$ different LD documents, each of which corresponds to one of the URIs in $\mathcal{U}_{\mathsf{TMsteps}}$ (and, thus, to a particular step in a particular computation of a particular TM). Mapping $adoc_{\mathsf{TMs}}$ is bijective and maps each $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$ to the corresponding $d_i^{w,x} \in D_{\mathsf{TMs}}$ ($\mathrm{dom}(adoc_{\mathsf{TMs}}) = \mathcal{U}_{\mathsf{TMsteps}}$). We emphasize that mapping $adoc_{\mathsf{TMs}}$ is (Turing) computable because a universal TM may determine by simulation whether the computation of a particular TM on a particular input halts before a particular number of steps (i.e. whether the i-th step in computation $c^{w,x}$ for a given URI $u_i^{w,x}$ may actually exist). Finally, mapping $data_{\mathsf{TMs}}$ is defined as follows: The set $data_{\mathsf{TMs}}\big(d_i^{w,x}\big)$ of RDF triples for an LD document $d_i^{w,x}$ is empty if computation $c^{w,x}$ does not halt with the i-th computation step. Otherwise, $data_{\mathsf{TMs}}\big(d_i^{w,x}\big)$ contains a single RDF triple $(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$ where $\mathsf{type} \in \mathcal{U}$ and $\mathsf{TerminatingComputation} \in \mathcal{U}$. Formally:

$$data_{\mathsf{TMs}}\big(d_i^{w,x}\big) = \begin{cases} \big\{(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})\big\} & \text{if computation } c^{w,x} \\ & \text{halts with the } i\text{-th} \\ & \text{step,} \\ \varnothing & \text{else.} \end{cases}$$

Mapping $data_{\mathsf{TMs}}$ is computable because a universal TM may determine by simulation whether the computation of a particular TM on a particular input halts after a given number of steps.

We now reduce the halting problem to TERMINATION(SPARQL$_{\mathsf{LD}}$). The input to the halting problem is a pair (w, x) consisting of a TM description w and a possible input word x. For the reduction we need a computable mapping f that, given such a pair (w, x), produces a tuple $(W, \mathcal{Q}^P)$ as input for TERMINATION(SPARQL$_{\mathsf{LD}}$). We define f as follows: Let (w, x) be an input to the halting problem, then $f(w, x) = \big(W_{\mathsf{TMs}}, \mathcal{Q}^{P_{w,x}}\big)$ with $P_{w,x} = (u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$. Given that W_{TMs} is independent of (w, x), it is easy to see that f is computable by TMs (including LD machines).

We emphasize that for any possible $\mathcal{Q}^{P_{w,x}}$ it holds:

$$\mathcal{Q}^{P_{w,x}}\big(W_{\mathsf{TMs}}\big) = \begin{cases} \{\mu_\varnothing\} & \text{if the computation of TM } M(w) \text{ on input } x \text{ halts,} \\ & \text{step,} \\ \varnothing & \text{else.} \end{cases}$$

where $\mu_\varnothing$ is the empty valuation with $\mathrm{dom}(\mu_\varnothing) = \varnothing$. Hence, any $\mathcal{Q}^{P_{w,x}}$ is satisfiable but not nontrivially satisfiable.

To show that TERMINATION(SPARQL$_{\mathsf{LD}}$) is not LD machine decidable, suppose it were LD machine decidable. In such a case an LD machine could answer the halting problem for any input (w, x) as follows: $M(w)$ halts on x if and only if an LD machine exists that computes $\mathcal{Q}^{P_{w,x}}(W_{\mathsf{TMs}}) = \{\mu_\varnothing\}$ and halts. However, we know the halting problem is undecidable for TMs (which includes LD machines). Hence, we have a contradiction and, thus, TERMINATION(SPARQL$_{\mathsf{LD}}$) cannot be LD machine decidable.

C.10 Proof of Proposition 2

Let $W = (D, data, adoc)$ be a finite Web of Linked Data. For each SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ it holds $\mathcal{Q}^P(W) = [\![P]\!]_{\mathrm{AllData}(W)}$ (cf. Definition 8). To prove the proposition it suffices to show $[\![P]\!]_{\mathrm{AllData}(W)}$ is finite for any possible SPARQL expression P. We use induction over the structure of SPARQL expressions for this proof:

Base case: Assume that SPARQL expression P is a triple pattern tp. In this case (cf. Definition 15)

$$[\![P]\!]_{\mathrm{AllData}(W)} = \big\{\mu \,\big|\, \mu \text{ is a valuation with } \mathrm{dom}(\mu) = \mathrm{vars}(tp) \text{ and } \mu[tp] \in \mathrm{AllData}(W)\big\}$$

Since W (and, thus, D) is finite and for all $d \in D$ the set $data(d)$ is finite, there exist only a finite number of RDF triples in $\mathrm{AllData}(W) = \bigcup_{d \in D} data(d)$. Hence, there can be only a finite number of different valuations μ with $\mu[tp] \in \mathrm{AllData}(W)$ and, thus, $[\![P]\!]_{\mathrm{AllData}(W)}$ must be finite.

Induction step: Our inductive hypothesis is that for SPARQL expressions P_1 and P_2, $[\![P_1]\!]_{\mathrm{AllData}(W)}$ and $[\![P_2]\!]_{\mathrm{AllData}(W)}$ is finite, respectively. Based on this hypothesis we show that for any SPARQL expression P that can be constructed using P_1 and P_2 (cf. Definition 14), it holds $[\![P]\!]_{\mathrm{AllData}(W)}$ is finite. According to Definition 14 we distinguish the following cases:

- P is $(P_1 \text{ AND } P_2)$. In this case $[\![P]\!]_{\mathrm{AllData}(W)} = [\![P_1]\!]_{\mathrm{AllData}(W)} \bowtie [\![P_2]\!]_{\mathrm{AllData}(W)}$. The result of the join may contain at most $\big|[\![P_1]\!]_{\mathrm{AllData}(W)}\big| \cdot \big|[\![P_2]\!]_{\mathrm{AllData}(W)}\big|$ elements, which is a finite number because $[\![P_1]\!]_{\mathrm{AllData}(W)}$ and $[\![P_2]\!]_{\mathrm{AllData}(W)}$ are finite.
- P is $(P_1 \text{ UNION } P_2)$. In this case $[\![P]\!]_{\mathrm{AllData}(W)} = [\![P_1]\!]_{\mathrm{AllData}(W)} \cup [\![P_2]\!]_{\mathrm{AllData}(W)}$. The result of the union may contain at most $\big|[\![P_1]\!]_{\mathrm{AllData}(W)}\big| + \big|[\![P_2]\!]_{\mathrm{AllData}(W)}\big|$ elements, which is a finite number because $[\![P_1]\!]_{\mathrm{AllData}(W)}$ and $[\![P_2]\!]_{\mathrm{AllData}(W)}$ are finite.
- P is $(P_1 \text{ OPT } P_2)$. In this case $[\![P]\!]_{\mathrm{AllData}(W)} = [\![P_1]\!]_{\mathrm{AllData}(W)} ⟕ [\![P_2]\!]_{\mathrm{AllData}(W)}$. The result of the left outer join contains at most $\big|[\![P_1]\!]_{\mathrm{AllData}(W)}\big| \cdot \big|[\![P_2]\!]_{\mathrm{AllData}(W)}\big|$ elements, which is a finite number because $[\![P_1]\!]_{\mathrm{AllData}(W)}$ and $[\![P_2]\!]_{\mathrm{AllData}(W)}$ are finite.
- P is $(P_1 \text{ FILTER } R)$. In this case $[\![P]\!]_{\mathrm{AllData}(W)} = \sigma_R\big([\![P_1]\!]_{\mathrm{AllData}(W)}\big)$. The result of the selection may contain at most $\big|[\![P_1]\!]_{\mathrm{AllData}(W)}\big|$ elements, which is a finite number because $[\![P_1]\!]_{\mathrm{AllData}(W)}$ is finite.

C.11 Proof of Theorem 3

We formally define the finiteness problem for SPARQL$_{\mathsf{LD}}$ as follows:

Problem:	FINITENESS(SPARQL$_{\mathsf{LD}}$)
Web Input:	a (potentially infinite) Web of Linked Data W
Ordinary Input:	a satisfiable SPARQL expression P
Question:	Is the result of (the satisfiable) SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ over W finite?

We show that FINITENESS(SPARQL$_{\mathsf{LD}}$) is not LD machine decidable by reducing the halting problem to FINITENESS(SPARQL$_{\mathsf{LD}}$). While this proof is similar to the proof of Theorem 2 (cf. Section C.9), we now use a Web of Linked Data W_{TMs2} which differs from W_{TMs} in the way it describes all possible computations of all Turing machines (TM).

For the proof we use the same symbols as in Section C.9. That is, $\mathcal{W}$ denotes the countably infinite set of all words that describe TMs. $M(w)$ denote the machine described by w (for all $w \in \mathcal{W}$); $c^{w,x}$ denotes the computation of $M(w)$ on input x; $u_i^{w,x} \in \mathcal{U}$ identifies the i-th step in computation $c^{w,x}$. The set of all these identifiers is denoted by $\mathcal{U}_{\mathsf{TMsteps}}$ (recall that, although $\mathcal{U}_{\mathsf{TMsteps}}$ is infinite, it is countable).

We now define W_{TMs2} as a Web of Linked Data $(D_{\mathsf{TMs2}}, data_{\mathsf{TMs2}}, adoc_{\mathsf{TMs2}})$ similar to the Web W_{TMs} used in Section C.9: D_{TMs2} and $adoc_{\mathsf{TMs2}}$ are the same is in W_{TMs}. That is, D_{TMs2} consists of $|\mathcal{U}_{\mathsf{TMsteps}}|$ different LD documents, each of which corresponds to one of the URIs in $\mathcal{U}_{\mathsf{TMsteps}}$. Mapping $adoc_{\mathsf{TMs2}}$ is bijective and maps each $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$ to the corresponding $d_i^{w,x} \in D_{\mathsf{TMs2}}$. Mapping $data_{\mathsf{TMs2}}$ for W_{TMs2} is different from the corresponding mapping for W_{TMs}: The set $data_{\mathsf{TMs2}}(d_i^{w,x})$ of RDF triples for *each* LD document $d_i^{w,x}$ contains a single RDF triple $(u_i^{w,x}, \mathsf{first}, u_1^{w,x})$ which associates the corresponding computation step $u_i^{w,x}$ with the first step of the corresponding computation $c^{w,x}$ ($\mathsf{first} \in \mathcal{U}$ denotes a URI for this relationship).

Before we come to the reduction we highlight a property of W_{TMs2} that is important for our proof. Each RDF triple $(u_i^{w,x}, \mathsf{first}, u_1^{w,x})$ establishes a data link from $d_i^{w,x}$ to $d_1^{w,x}$. Hence, the link graph of W_{TMs2} consists of an infinite number of separate subgraphs, each of which corresponds to a particular computation $c^{w,x}$, is weakly connected, and has a star-like form where the corresponding $d_1^{w,x}$ is in the center of the star. More precisely, for subgraph $(V^{w_j,x_k}, E^{w_j,x_k})$ that corresponds to computation c^{w_j,x_k} it holds

$$V^{w_j,x_k} = \left\{ d_i^{w,x} \in D_{\mathsf{TMs2}} \,\middle|\, w = w_j \text{ and } x = x_k \right\}$$

and

$$E^{w_j,x_k} = V^{w_j,x_k} \times \left\{ d_1^{w_j,x_k} \right\}.$$

Each of these subgraphs is infinitely large (i.e. has an infinite number of vertices) if and only if the corresponding computation halts.

For the reduction we use mapping f which is defined as follows: Let w be the description of a TM, let x be a possible input word for $M(w)$, and let $?v \in \mathcal{V}$ be a query variable, then $f(w,x) = \left(W_{\mathsf{TMs2}}, P_{w,x}\right)$ with $P_{w,x} = (?v, \mathsf{first}, u_1^{w,x})$. Given

that W_{TMs2} is independent of (w, x), it is easy to see that f is computable by TMs (including LD machines).

To show that FINITENESS(SPARQL$_{\mathsf{LD}}$) is not LD machine decidable, suppose it were LD machine decidable. In such a case an LD machine could answer the halting problem for any input (w, x) as follows: $M(w)$ halts on x if and only if $\mathcal{Q}^{P_{w,x}}(W_{\mathsf{TMs2}})$ is finite. However, we know the halting problem is undecidable for TMs (which includes LD machines). Hence, we have a contradiction and, thus, FINITENESS(SPARQL$_{\mathsf{LD}}$) cannot be LD machine decidable.

C.12 Proof of Proposition 3, Case 1

Let:

- $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that is monotonic;
- $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses the same SPARQL expression P as $\mathcal{Q}^P$ (and an arbitrary reachability criterion c and (finite) set $S \subset \mathcal{U}$ of seed URIs);
- W be a Web of Linked Data; and
- $W_c^{(S,P)}$ denote the (S, c, P)-reachable part of W.

W.l.o.g. it suffices to show: $\mathcal{Q}_c^{P,S}(W) \subseteq \mathcal{Q}^P(W)$.

Since $W_c^{(S,P)}$ is an induced subweb of W (cf. Definition 11) and $\mathcal{Q}^P$ is monotonic, it holds $\mathcal{Q}^P(W_c^{(S,P)}) \subseteq \mathcal{Q}^P(W)$. Furthermore, we have $\mathcal{Q}_c^{P,S}(W) = \mathcal{Q}^P(W_c^{(S,P)})$ (cf. Proposition 3, case 2). Hence, $\mathcal{Q}_c^{P,S}(W) \subseteq \mathcal{Q}^P(W)$.

C.13 Proof of Proposition 3, Case 2

Let:

- $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query (that uses SPARQL expression P, reachability criterion c, and (finite) set $S \subset \mathcal{U}$ of seed URIs);
- $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that uses the same SPARQL expression P as $\mathcal{Q}_c^{P,S}$;
- W be an arbitrary Web of Linked Data; and
- $W_c^{(S,P)}$ denote the (S, c, P)-reachable part of W.

It holds:

- $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathrm{AllData}(W_c^{(S,P)})}$ (cf. Definition 12) and
- $\mathcal{Q}^P(W_c^{(S,P)}) = [\![P]\!]_{\mathrm{AllData}(W_c^{(S,P)})}$ (cf. Definition 8).

Hence, $\mathcal{Q}_c^{P,S}(W) = \mathcal{Q}^P(W_c^{(S,P)})$.

C.14 Proof of Proposition 4

Let:

- $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query (that uses SPARQL expression P, reachability criterion c, and (finite) set $S \subset \mathcal{U}$ of seed URIs);
- $W = (D, data, adoc)$ be a *finite* Web of Linked Data; and
- $W_c^{(S,P)} = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ be the (S, c, P)-reachable part of W.

W.l.o.g. it suffices to show: $\mathcal{Q}_c^{P,S}\big(W\big)$ is finite and $W_c^{(S,P)}$ is finite.

W is finite, which means D is finite. Therefore, any subset of D must also be finite; this includes $D_{\mathfrak{R}} \subseteq D$ because $W_c^{(S,P)}$ is an induced subweb of W (cf. Definition 11). Hence, $W_c^{(S,P)}$ is finite.

The finiteness of $\mathcal{Q}_c^{P,S}\big(W\big)$ follows directly from the finiteness of $W_c^{(S,P)}$ (and is independent of the finiteness of W) as the following lemma shows.

Lemma 7. *For any SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and any (potentially infinite) Web of Linked Data W it holds: If $W_c^{(S,P)}$ is finite, then $\mathcal{Q}_c^{P,S}\big(W\big)$ is finite.*

Proof of Lemma 7. The lemma follows directly from Propositions 3 (case 2) and 2. □

C.15 Proof of Proposition 5

Let:

- P be a SPARQL expression;
- c and c' be reachability criteria;
- $S \subset \mathcal{U}$ be a finite but nonempty set of seed URIs;
- $W = (D, data, adoc)$ be an *infinite* Web of Linked Data.

1. $W_{c_{\mathsf{None}}}^{(S,P)}$ is always finite; so is $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}\big(W\big)$.
Let $D_{\mathfrak{R}}$ denote the set of all LD documents in $W_{c_{\mathsf{None}}}^{(S,P)}$. Since c_{None} always returns false it is easily verified that there is no LD document $d \in D$ that satisfies case 2 in Definition 10. Hence, it must hold $D_{\mathfrak{R}} = \{d \in D \,|\, u \in S \text{ and } adoc(u) = d\}$ (cf. case 1 in Definition 10). Since S is finite we see that $D_{\mathfrak{R}}$ is guaranteed to be finite (and so is $W_{c_{\mathsf{None}}}^{(S,P)}$). The finiteness of $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}\big(W\big)$ can then be shown using Lemma 7 (cf. Section C.14).

2. If $W_c^{(S,P)}$ is finite, then $\mathcal{Q}_c^{P,S}\big(W\big)$ is finite.
See Lemma 7 in Section C.14.

3. If $\mathcal{Q}_c^{P,S}\big(W\big)$ is infinite, then $W_c^{(S,P)}$ is infinite.
Let $\mathcal{Q}_c^{P,S}\big(W\big)$ be infinite. We prove by contradiction that $W_c^{(S,P)}$ is infinite: Suppose $W_c^{(S,P)}$ were finite. In this case $\mathcal{Q}_c^{P,S}\big(W\big)$ would be finite (cf. Lemma 7 in Section C.14), which is a contradiction to our premise. Hence, $W_c^{(S,P)}$ must be infinite.

4. If c is less restrictive than c' and $W_c^{(S,P)}$ is finite, then $W_{c'}^{(S,P)}$ is finite.
If $W_c^{(S,P)}$ is finite, then exist finitely many LD documents $d \in D$ that are (c, P)-reachable from S in W. A subset of them is also (c', P)-reachable from S in W because c is less restrictive than c'. Hence, $W_{c'}^{(S,P)}$ must also be finite.

5. If c' is less restrictive than c and $W_c^{(S,P)}$ is infinite, then $W_{c'}^{(S,P)}$ is infinite.
If $W_c^{(S,P)}$ is infinite, then exist infinitely many LD documents $d \in D$ that are (c, P)-reachable from S in W. Each of them is also (c', P)-reachable from S in W because c' is less restrictive than c. Hence, $W_{c'}^{(S,P)}$ must be infinite.

C.16 Proof of Theorem 4

We formally define the decision problems FINITENESSREACHABLEPART and FINITENESS($\text{SPARQL}_{\text{LD(R)}}$) as follows:

Problem:	FINITENESSREACHABLEPART
Web Input:	a (potentially infinite) Web of Linked Data W
Ordinary Input:	a finite but nonempty set $S \subset \mathcal{U}$ of seed URIs
	a reachability criterion c that is less restrictive than c_{None}
	a SPARQL expression P
Question:	Is the (S, c, P)-reachable part of W finite?

Problem:	FINITENESS($\text{SPARQL}_{\text{LD(R)}}$)
Web Input:	a (potentially infinite) Web of Linked Data W
Ordinary Input:	a finite but nonempty set $S \subset \mathcal{U}$ of seed URIs
	a reachability criterion c that is less restrictive than c_{None}
	a SPARQL expression P
Question:	Is the result of $\text{SPARQL}_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ over W finite?

We now prove Theorem 4 by reducing the halting problem to FINITENESSREACHABLEPART and FINITENESS($\text{SPARQL}_{\text{LD(R)}}$). While this proof is similar to the proofs of Theorem 2 (cf. Section C.9) and Theorem 3 (cf. Section C.11), we now use a Web of Linked Data W_{TMs3} which (again) differs from W_{TMs} and W_{TMs2} in the way it describes all possible computations of all Turing machines (TM).

For the proof we use the same symbols as in Section C.9. That is, $\mathcal{W}$ denotes the countably infinite set of all words that describe TMs. $M(w)$ denote the machine described by w (for all $w \in \mathcal{W}$); $c^{w,x}$ denotes the computation of $M(w)$ on input x; $u_i^{w,x} \in \mathcal{U}$ identifies the i-th step in computation $c^{w,x}$. The set of all these identifiers is denoted by $\mathcal{U}_{\mathsf{TMsteps}}$ (recall that, although $\mathcal{U}_{\mathsf{TMsteps}}$ is infinite, it is countable).

We now define W_{TMs3} as a Web of Linked Data $(D_{\mathsf{TMs3}}, data_{\mathsf{TMs3}}, adoc_{\mathsf{TMs3}})$ similar to the Web W_{TMs} used in Section C.9: D_{TMs3} and $adoc_{\mathsf{TMs3}}$ are the same is in W_{TMs}. That is, D_{TMs3} consists of $|\mathcal{U}_{\mathsf{TMsteps}}|$ different LD documents, each of which corresponds to one of the URIs in $\mathcal{U}_{\mathsf{TMsteps}}$. Mapping $adoc_{\mathsf{TMs3}}$ is bijective and maps each $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$ to the corresponding $d_i^{w,x} \in D_{\mathsf{TMs3}}$. Mapping $data_{\mathsf{TMs3}}$ for W_{TMs3} is different from the corresponding mapping for W_{TMs}: The set $data_{\mathsf{TMs3}}(d_i^{w,x})$ of RDF

triples for an LD document $d_i^{w,x}$ is empty if computation $c^{w,x}$ halts with the i-th computation step. Otherwise, $data_{\mathsf{TMs3}}(d_i^{w,x})$ contains a single RDF triple $(u_i^{w,x}, \mathsf{next}, u_{i+1}^{w,x})$ which associates the computation step $u_i^{w,x}$ with the next step in $c^{w,x}$ ($\mathsf{next} \in \mathcal{U}$ denotes a URI for this relationship). Formally:

$$data_{\mathsf{TMs3}}(d_i^{w,x}) = \begin{cases} \varnothing & \text{if computation } c^{w,x} \text{ halts} \\ & \text{with the } i\text{-th step,} \\ \{(u_i^{w,x}, \mathsf{next}, u_{i+1}^{w,x})\} & \text{else.} \end{cases}$$

Mapping $data_{\mathsf{TMs3}}$ is (Turing) computable because a universal TM may determine by simulation whether the computation of a particular TM on a particular input halts after a given number of steps.

Before we come to the reduction we highlight a property of W_{TMs3} that is important for our proof. Each RDF triple $(u_i^{w,x}, \mathsf{next}, u_{i+1}^{w,x})$ establishes a data link from $d_i^{w,x}$ to $d_{i+1}^{w,x}$. Due to such links we recursively may reach all LD documents about all steps in a particular computation of any TM. Hence, for each possible computation $c^{w,x}$ of any TM $M(w)$ we have a (potentially infinite) simple path $(d_1^{w,x}, \ldots, d_i^{w,x}, \ldots)$ in the link graph of W_{TMs3}. Each of these paths is finite if and only if the corresponding computation halts. Finally, we note that each of these paths forms a separate subgraph of the link graph of W_{TMs3} because we use a separate set of step URIs for each computation and the RDF triples in the corresponding LD documents only mention steps from the same computation.

For the reduction we use mapping f which is defined as follows: Let w be the description of a TM, let x be a possible input word for $M(w)$, and let $?a, ?b \in \mathcal{V}$ be two distinct query variables, then $f(w,x) = (W_{\mathsf{TMs3}}, S_{w,x}, c_{\mathsf{Match}}, P_{w,x})$ with $S_{w,x} = \{u_1^{w,x}\}$ and $P_{w,x} = (?a, \mathsf{next}, ?b)$. Given that c_{Match} and W_{TMs3} are independent of (w,x), it can be easily seen that f is computable by TMs (including LD machines).

To show that FINITENESSREACHABLEPART is not LD machine decidable, suppose it were LD machine decidable. In such a case an LD machine could answer the halting problem for any input (w,x) as follows: $M(w)$ halts on x if and only if the $(S_{w,x}, c_{\mathsf{Match}}, P_{w,x})$-reachable part of W_{TMs3} is finite. However, we know the halting problem is undecidable for TMs (which includes LD machines). Hence, we have a contradiction and, thus, FINITENESSREACHABLEPART cannot be LD machine decidable.

The proof that FINITENESS($\text{SPARQL}_{\mathsf{LD(R)}}$) is undecidable is similar to that for FINITENESSREACHABLEPART. Hence, we only outline the idea: Instead of reducing the halting problem to FINITENESSREACHABLEPART based on mapping f we now reduce the halting problem to FINITENESSQUERYRESULT using the same mapping. If FINITENESS($\text{SPARQL}_{\mathsf{LD(R)}}$) were decidable then we could answer the halting problem for any (w,x): $M(w)$ halts on x if and only if $\mathcal{Q}_{c_{\mathsf{Match}}}^{P_{w,x}, S_{w,x}}(W_{\mathsf{TMs3}})$ is finite.

C.17 Proof of Proposition 6, Case 1

This proof is similar to the proof of Proposition 1, case 1 (cf. Section C.2).

If: Let P be a SPARQL expression that is satisfiable and let $\mathcal{Q}_c^{P,S}$ be an arbitrary $\text{SPARQL}_{\mathsf{LD(R)}}$ query that uses P, a nonempty set $S \subset \mathcal{U}$ of seed URIs, and an arbitrary reachability criterion c. W.l.o.g. it suffices to show that $\mathcal{Q}_c^{P,S}$ is satisfiable.

For this proof we use the notion of (P, G)-lineage of valuations that we introduced before (cf. Definition 16 in Section C.2). Recall that for any SPARQL expression P, any (potentially infinite) set G of RDF triples, and any valuation $\mu \in [\![P]\!]_G$ it holds: i) $G' = \mathrm{lin}^{P,G}(\mu)$ is finite and ii) $\mu \in [\![P]\!]_{G'}$.

Due to the satisfiability of P exists a set of RDF triples G such that $[\![P]\!]_G \neq \varnothing$. W.l.o.g., let μ be an arbitrary solution for P in G, that is, $\mu \in [\![P]\!]_G$. Furthermore, let $G' = \mathrm{lin}^{P,G}(\mu)$ be the (P, G)-lineage of μ. We use G' to construct a Web of Linked Data $W_\mu = (D_\mu, data_\mu, adoc_\mu)$ which consists of a single LD document. This document may be retrieved using any URI from the (nonempty) set S and it contains the (P, G)-lineage of μ (recall that the lineage is guaranteed to be a finite). Formally:

$$D_\mu = \{d\} \qquad data_\mu(d) = G' \qquad \forall\, u \in S : adoc_\mu(u) = d$$

Due to our construction it holds $\mathrm{AllData}(W_\mu) = \mathrm{AllData}(W_\mathfrak{R}) = G'$ where $W_\mathfrak{R}$ denotes the (S, c, P)-reachable part of W_μ. Thus, we have $\mathcal{Q}_c^{P,S}(W_\mu) = [\![P]\!]_{G'}$ (cf. Definition 12). Since we know $\mu \in [\![P]\!]_{G'}$ it holds $\mathcal{Q}_c^{P,S}(W_\mu) \neq \varnothing$, which shows that $\mathcal{Q}_c^{P,S}$ is satisfiable.

Only if: Let $\mathcal{Q}_c^{P,S}$ be a satisfiable SPARQL$_{\mathsf{LD(R)}}$ query that uses SPARQL expression P, a nonempty set $S \subset \mathcal{U}$ of seed URIs, and an arbitrary reachability criterion c. Since $\mathcal{Q}_c^{P,S}$ is satisfiable, exists a Web of Linked Data W such that $\mathcal{Q}_c^{P,S}(W) \neq \varnothing$. According to Definition 12 we also have $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathrm{AllData}(W)}$. Thus, we conclude that P is satisfiable.

C.18 Proof of Proposition 6, Case 2

This proof is similar to the proof of Proposition 1, case 2 (cf. Section C.3).

If: Let P be a SPARQL expression that is nontrivially satisfiable and let $\mathcal{Q}_c^{P,S}$ be an arbitrary SPARQL$_{\mathsf{LD(R)}}$ query that uses P, a nonempty set $S \subset \mathcal{U}$ of seed URIs, and an arbitrary reachability criterion c. W.l.o.g. it suffices to show that $\mathcal{Q}_c^{P,S}$ is nontrivially satisfiable.

Due to the nontrivial satisfiability of P exists a set of RDF triples G and a valuation μ such that i) $\mu \in [\![P]\!]_G$ and ii) $\mathrm{dom}(\mu) \neq \varnothing$. Let $G' = \mathrm{lin}^{P,G}(\mu)$ be the (P, G)-lineage of μ. We use G' to construct a Web of Linked Data $W_\mu = (D_\mu, data_\mu, adoc_\mu)$ which consists of a single LD document. This document may be retrieved using any URI from the (nonempty) set S and it contains the (P, G)-lineage of μ (recall that the lineage is guaranteed to be a finite). Formally:

$$D_\mu = \{d\} \qquad data_\mu(d) = G' \qquad \forall\, u \in S : adoc_\mu(u) = d$$

Due to our construction it holds $\mathrm{AllData}(W_\mu) = \mathrm{AllData}(W_\mathfrak{R}) = G'$ where $W_\mathfrak{R}$ denotes the (S, c, P)-reachable part of W_μ. Thus, we have $\mathcal{Q}_c^{P,S}(W_\mu) = [\![P]\!]_{G'}$ (cf. Definition 12). Since we know $\mu \in [\![P]\!]_{G'}$ and $\mathrm{dom}(\mu) \neq \varnothing$, we conclude that $\mathcal{Q}_c^{P,S}$ is nontrivially satisfiable.

Only if: Let $\mathcal{Q}_c^{P,S}$ be a nontrivially satisfiable SPARQL$_{\mathsf{LD(R)}}$ query that uses SPARQL expression P, a nonempty set $S \subset \mathcal{U}$, and an arbitrary reachability criterion c. Since

$\mathcal{Q}_c^{P,S}$ is nontrivially satisfiable, exists a Web of Linked Data W and a valuation μ such that i) $\mu \in \mathcal{Q}_c^{P,S}(W)$ and ii) $\mathrm{dom}(\mu) \neq \varnothing$. According to Definition 12 we also have $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathrm{AllData}(W)}$. Thus, we conclude that P is nontrivially satisfiable.

C.19 Proof of Proposition 6, Case 3

If: Let:

- P be a SPARQL expression that is monotonic;
- $\mathcal{Q}_c^{P,S}$ be an arbitrary SPARQL$_{\mathsf{LD(R)}}$ query that uses P, a nonempty set $S \subset \mathcal{U}$ of seed URIs, and an arbitrary reachability criterion c; and
- W_1, W_2 be an arbitrary pair of Webs of Linked Data such that W_1 is an induced subweb of W_2; and
- $W_{\mathfrak{R}1} = (D_{\mathfrak{R}1}, data_{\mathfrak{R}1}, adoc_{\mathfrak{R}1})$ and $W_{\mathfrak{R}2} = (D_{\mathfrak{R}2}, data_{\mathfrak{R}2}, adoc_{\mathfrak{R}2})$ denote the (S, c, P)-reachable part of W_1 and of W_2, respectively.

To prove that $\mathcal{Q}_c^{P,S}$ is monotonic it suffices to show $\mathcal{Q}_c^{P,S}(W_1) \subseteq \mathcal{Q}_c^{P,S}(W_2)$.

Any LD document that is (c, P)-reachable from S in W_1 is also (c, P)-reachable from S in W_2 because W_1 is an induced subweb of W_2. Hence, $D_{\mathfrak{R}1} \subseteq D_{\mathfrak{R}2}$ and, thus, $\mathrm{AllData}(W_{\mathfrak{R}1}) \subseteq \mathrm{AllData}(W_{\mathfrak{R}2})$. Furthermore, $\mathcal{Q}_c^{P,S}(W_1) = [\![P]\!]_{\mathrm{AllData}(W_{\mathfrak{R}1})}$ and $\mathcal{Q}_c^{P,S}(W_2) = [\![P]\!]_{\mathrm{AllData}(W_{\mathfrak{R}2})}$ (cf. Definition 12). Due to the monotonicity of P it also holds $[\![P]\!]_{\mathrm{AllData}(W_{\mathfrak{R}1})} \subseteq [\![P]\!]_{\mathrm{AllData}(W_{\mathfrak{R}2})}$. Hence, $\mathcal{Q}_c^{P,S}(W_1) \subseteq \mathcal{Q}_c^{P,S}(W_2)$.

C.20 Proof of Proposition 7

Let:

- $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query (under c_{None}-semantics) such that $|S| = 1$;
- $W_1 = (D_1, data_1, adoc_1)$ and $W_2 = (D_2, data_2, adoc_2)$ be two Webs of Linked Data such that W_1 is an induced subweb of W_2; and
- $W_1^{\mathcal{R}}$ and $W_2^{\mathcal{R}}$ denote the $(S, c_{\mathsf{None}}, P)$-reachable part of W_1 and of W_2, respectively.

W.l.o.g. it suffices to show $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) \subseteq \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2)$. We distinguish the following three cases for $u \in S = \{u\}$:

1. $u \notin \mathrm{dom}(adoc_1)$ and $u \notin \mathrm{dom}(adoc_2)$.
   In this case $W_1^{\mathcal{R}}$ and $W_2^{\mathcal{R}}$ are equal to the empty Web (which contains no LD documents). Hence, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) = \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2) = \varnothing$.
2. $u \notin \mathrm{dom}(adoc_1)$ and $adoc_2(u) = d$ where $d \in D_2$.
   In this case $W_1^{\mathcal{R}}$ is equal to the empty Web, whereas $W_2^{\mathcal{R}}$ contains a single LD document, namely d. Hence, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) = \varnothing$ and $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2) = [\![P]\!]_{\mathrm{AllData}(W_2^{\mathcal{R}})} = [\![P]\!]_{data_2(d)}$ and, thus, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) \subseteq \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2)$.
3. $adoc_1(u) = d$ and $adoc_2(u) = d$ where $d \in D_1 \subseteq D_2$.
   In this case both Webs, $W_1^{\mathcal{R}}$ and $W_2^{\mathcal{R}}$, contain a single LD document, namely d. Hence, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) = \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2) = [\![P]\!]_{data_2(d)}$ (recall, in this case holds $data_1(d) = data_2(d)$).
4. $u \in \mathrm{dom}(adoc_1)$ and $u \notin \mathrm{dom}(adoc_2)$.
   This case is impossible because W_1 is an induced subweb of W_2.

Algorithm 2 The program of the $(P, S, c)'$-machine.

1: Call `lookup` for each $u \in S$.

2: $expansionCompleted := \text{false}$
3: **while** $expansionCompleted = \text{false}$ **do**
4: Scan the link traversal tape for an RDF triple t and a URI $u \in \text{uris}(t)$ such that i) $c(t, u, P) = \text{true}$ and ii) the word on the link traversal tape neither contains $\text{enc}(u)\,\text{enc}(adoc(u))\,\sharp$ nor $\text{enc}(u)\,\sharp$. If such t and u exist, call `lookup` for u; otherwise $expansionCompleted := \text{true}$.
5: **end while**

6: Let G denote the set of all RDF triples currently encoded on the link traversal tape. For each $\mu \in [\![P]\!]_G$ add $\text{enc}(\mu)$ to the output.

C.21 Proof of Lemma 2

For proving Lemma 2 we introduce specific LD machines for SPARQL$_{\text{LD(R)}}$ queries. We call these machines $(P, S, c)'$-machines. The $(P, S, c)'$-machine for SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ implements a generic (i.e. input independent) computation of $\mathcal{Q}_c^{P,S}$. For any nontrivially satisfiable SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ we shall see that if and only if the (S, c, P)-reachable part of an arbitrary Web of Linked Data W is finite, the corresponding $(P, S, c)'$-machine computes $\mathcal{Q}_c^{P,S}(W)$ and halts. Formally, we define $(P, S, c)'$-machines as follows:

Definition 18. *Let $S \subset \mathcal{U}$ be a finite set of seed URIs; let c be a reachability criterion; and let P be a SPARQL expression. The $(P, S, c)'$-**machine** is an LD machine that implements Algorithm 2. This algorithm makes use of a subroutine called `lookup`. This subroutine, when called with a URI $u \in \mathcal{U}$, i) writes $\text{enc}(u)$ to the right end of the word on the link traversal tape, ii) enters the expand state, and iii) performs the expand procedure as specified in Definition 4.*

Before we complete the proof of Lemma 2 we discuss properties of any $(P, S, c)'$-machine as they are relevant for the proof. The computation of each $(P, S, c)'$-machine (with a Web of Linked Data W encoded on its input tape) starts with an initialization (cf. line 1 in Algorithm 2). After the initialization, the machine enters a (potentially non-terminating) loop that recursively discovers (i.e. expands) all LD documents of the corresponding reachable part of W. The following lemma shows that for each such document exists an iteration of the loop during which the machine copies that document to its link traversal tape.

Lemma 8. *Let:*

- *$M^{(P,S,c)'}$ be the $(P, S, c)'$-machine for a SPARQL expression P, a finite set $S \subset \mathcal{U}$, and a reachability criterion c;*
- *$W = (D, data, adoc)$ be a (potentially infinite) Web of Linked Data encoded on the Web tape of $M^{(P,S,c)'}$; and*
- *$d \in D$ be an LD document that is (c, P)-reachable from S in W.*

During the execution of Algorithm 2 by $M^{(P,S,c)'}$ exists an iteration of the loop (lines 3 to 5) after which the word on the link traversal tape of $M^{(P,S,c)'}$ (permanently) contains $\mathrm{enc}(d)$.

Proof of Lemma 8. To prove the lemma we first emphasize that $M^{(P,S,c)'}$ only appends to the word on its link traversal tape. Hence, $M^{(P,S,c)'}$ never removes $\mathrm{enc}(d)$ from that word once it has been added. The same holds for the encoding of any other LD document $d' \in D$.

Since d is (c, P)-reachable from S in W, the link graph for W contains at least one finite path $(d_0, \ldots, d_n)$ of LD documents d_i where i) $n \in \{0, 1, \ldots\}$, ii) $d_n = d$, iii) $\exists\, u \in S : adoc(u) = d_0$, and iv) for each $i \in \{1, \ldots, n\}$ it holds:

$$\exists\, t \in data(d_{i-1}) : \Big(\exists\, u \in \mathrm{uris}(t) : \big(adoc(u) = d_i \text{ and } c(t, u, P) = \mathrm{true}\big)\Big) \qquad (1)$$

Let $(d_0^*, \ldots, d_n^*)$ be such a path. We use this path to prove the lemma. More precisely, we show by induction over $i \in \{0, \ldots, n\}$ that there exists an iteration after which the word on the link traversal tape of $M^{(P,S,c)'}$ contains $\mathrm{enc}(d_n^*)$ (which is the same as $\mathrm{enc}(d)$ because $d_n^* = d$).

Base case ($i = 0$): Since $\exists\, u \in S : adoc(u) = d_0^*$ it is easy to verify that after the 0-th iteration (i.e. before the first iteration) the word on the link traversal tape of $M^{(P,S,c)'}$ contains $\mathrm{enc}(d_0^*)$ (cf. line 1 in Algorithm 2).

Induction step ($i > 0$): Our inductive hypothesis is: There exists an iteration after which the word on the link traversal tape of $M^{(P,S,c)'}$ contains $\mathrm{enc}(d_{i-1}^*)$. Let this be the j-th iteration. Based on the hypothesis we show that there exists an iteration after which the word on the link traversal tape of $M^{(P,S,c)'}$ contains $\mathrm{enc}(d_i^*)$. We distinguish two cases: after the j-th iteration the word on the link traversal tape either already contains $\mathrm{enc}(d_i^*)$ or it does not contain $\mathrm{enc}(d_i^*)$. We have to discuss the latter case only. Due to (1) exist $t^* \in data(d_{i-1}^*)$ and $u^* \in \mathrm{uris}(t^*)$ such that $adoc(u^*) = d_i^*$ and $c(t^*, u^*, P) = \mathrm{true}$. Hence, there exists a $\delta \in \mathbb{N}^+$ such that $M^{(P,S,c)'}$ finds t^* and u^* in the (j+δ)-th iteration. Since $M^{(P,S,c)'}$ calls `lookup` for u^* in that iteration (cf. line 4 in Algorithm 2), the link traversal tape contains $\mathrm{enc}(d_i^*)$ after that iteration. □

While Lemma 8 shows that Algorithm 2 discovers all reachable LD documents, the following lemma verifies that the algorithm does not copy data from unreachable documents to the link traversal tape.

Lemma 9. *Let:*

- *$M^{(P,S,c)'}$ be the $(P, S, c)'$-machine for a SPARQL expression P, a finite set $S \subset \mathcal{U}$, and a reachability criterion c;*
- *$W = (D, data, adoc)$ be a (potentially infinite) Web of Linked Data encoded on the Web tape of $M^{(P,S,c)'}$; and*
- *$W_c^{(S,P)}$ denotes the (S, c, P)-reachable part of W.*

For any RDF triple t encoded on the link traversal tape of $M^{(P,S,c)'}$ it holds (at any point of the computation): $t \in \mathrm{AllData}(W_c^{(S,P)})$.

Proof of Lemma 9. Let w_j denote the word on the link traversal tape of $M^{(P,S,c)'}$ when $M^{(P,S,c)'}$ finishes the j-th iteration of the loop in Algorithm 2; w_0 denotes the corresponding word before the first iteration. To prove the lemma it is sufficient to show for each w_j (where $j \in \{0, 1, ...\}$) exists a finite sequence $u_1, ..., u_{n_j}$ of n_j different URIs $u_i \in \mathcal{U}$ (for all $i \in [1, n_j]$) such that i) w_j is[7]

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,...\,\sharp\,\mathrm{enc}(u_{n_j})\,\mathrm{enc}(adoc(u_{n_j}))\,\sharp$$

and ii) for each $i \in [1, n_j]$ either $u_i \notin \mathrm{dom}(adoc)$ (and, thus, $adoc(u_i)$ is undefined) or $adoc(u_i)$ is an LD document which is (c, P)-reachable from S in W. We use an induction over j for this proof.

Base case ($j = 0$): The computation of $M^{(P,S,c)'}$ starts with an empty link traversal tape (cf. Definition 4). Due to the initialization, w_0 is a concatenation of sub-words $\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ for all $u \in S$ (cf. line 1 in Algorithm 2). Hence, we have a corresponding sequence $u_1, ..., u_{n_0}$ where $n_0 = |S|$ and $\forall\, i \in [1, n_0] : u_i \in S$. The order of the URIs in that sequence depends on the order in which they have been looked up and is irrelevant for our proof. For all $u \in S$ it holds either $u_i \notin \mathrm{dom}(adoc)$ or $adoc(u)$ is (c, P)-reachable from S in W (cf. case 1 in Definition 10).

Induction step ($j > 0$): Our inductive hypothesis is that there exists a finite sequence $u_1, ..., u_{n_{j-1}}$ of n_{j-1} different URIs ($\forall\, i \in [1, n_{j-1}] : u_i \in \mathcal{U}$) such that i) w_{j-1} is

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,...\,\sharp\,\mathrm{enc}(u_{n_{j-1}})\,\mathrm{enc}(adoc(u_{n_{j-1}}))\,\sharp$$

and ii) for each $i \in [1, n_{j-1}]$ either $u_i \notin \mathrm{dom}(adoc)$ or $adoc(u_i)$ is (c, P)-reachable from S in W. In the j-th iteration $M^{(P,S,c)'}$ finds an RDF triple t encoded as part of w_{j-1} such that $\exists\, u \in \mathrm{uris}(t) : c(t, u, P) = \mathrm{true}$ and `lookup` has not been called for u. The machine calls `lookup` for u, which changes the word on the link traversal tape to w_j. Hence, w_j is equal to $w_{j-1}\,\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ and, thus, our sequence of URIs for w_j is $u_1, ..., u_{n_{j-1}}, u$. It remains to show that if $u \in \mathrm{dom}(adoc)$ then $adoc(u)$ is (c, P)-reachable from S in W.

Assume $u \in \mathrm{dom}(adoc)$. Since RDF triple t is encoded as part of w_{j-1} we know, from our inductive hypothesis, that t must be contained in the data of an LD document d^* that is (c, P)-reachable from S in W (and for which exists $i \in [1, n_{j-1}]$ such that $adoc(u_i) = d^*$). Therefore, t and u satisfy the requirements as given in case 2 of Definition 10 and, thus, $adoc(u)$ is (c, P)-reachable from S in W. □

After verifying that Algorithm 2 is sound (cf. Lemma 9) and complete (cf. Lemma 8) w.r.t. discovering reachable LD documents, we now show that an execution of the algorithm terminates if the corresponding reachable part of the input Web is finite.

Lemma 10. *Let:*

- *$M^{(P,S,c)'}$ be the $(P, S, c)'$-machine for a SPARQL expression P, a finite set $S \subset \mathcal{U}$, and a reachability criterion c;*
- *$W = (D, data, adoc)$ be a (potentially infinite) Web of Linked Data encoded on the Web tape of $M^{(P,S,c)'}$; and*

[7] We assume $\mathrm{enc}(adoc(u_i))$ is the empty word if $adoc(u_i)$ is undefined (i.e. $u_i \notin \mathrm{dom}(adoc)$).

– $W_c^{(S,P)}$ *denotes the* (S, c, P)*-reachable part of* W.

The computation of $M^{(P,S,c)'}$ *halts after a finite number of steps if* $W_c^{(S,P)}$ *is finite.*

Proof of Lemma 10. Let $W_c^{(S,P)}$ be finite. To show that the computation of $M^{(P,S,c)'}$ on (Web) input $\mathrm{enc}(W)$ halts after a finite number of steps we emphasize the following facts:

1. Each call of subroutine `lookup` by $M^{(P,S,c)'}$ terminates because the encoding of W is ordered following the order of the URIs in $\mathrm{dom}(adoc)$.
2. $M^{(P,S,c)'}$ completes the initialization in line 1 of Algorithm 2 after a finite number of steps because S is finite.
3. At any point in the computation the word on the link traversal tape of $M^{(P,S,c)'}$ is finite because $M^{(P,S,c)'}$ only gradually appends (encoded) LD documents to that tape (one document per iteration) and the encoding of each document is finite (recall that the set of RDF triples $data(d)$ for each LD document $d \in D$ is finite).
4. During each iteration of the loop in Algorithm 2, $M^{(P,S,c)'}$ completes the scan of its link traversal tape (cf. line 4) after a finite number of computation steps because the word on that tape is always finite. Thus, $M^{(P,S,c)'}$ finishes each iteration of the loop after a finite number of steps.
5. $M^{(P,S,c)'}$ considers only those URIs for a call of subroutine `lookup` that i) have not been considered before and that ii) are mentioned in (RDF triples of) LD documents from $W_c^{(S,P)}$ (cf. line 4). Since $W_c^{(S,P)}$ is finite there is only a finite number of such URIs and, thus, the loop in Algorithm 2 as performed by $M^{(P,S,c)'}$ has a finite number of iterations.
6. Due to the finiteness of the word on the link traversal tape the set G used in line 6 of Algorithm 2 is finite and, thus, $[\![P]\!]_G$ is finite. As a consequence $M^{(P,S,c)'}$ requires only a finite number of computation steps for executing line 6.

Altogether, these facts prove Lemma 10. □

We now prove Lemma 2. Let:

– $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that is nontrivially satisfiable;
– W be a (potentially infinite) Web of Linked Data; and
– $W_c^{(S,P)} = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ denote the (S, c, P)-reachable part of W.

If: Let $W_c^{(S,P)}$ be finite. We have to show that there exists an LD machine that computes $\mathcal{Q}_c^{P,S}(W)$ and halts after a finite number of computation steps. Based on Lemmas 8 to 10 it is easy to verify that the $(P, S, c)'$-machine (for P, S and c as used by $\mathcal{Q}_c^{P,S}$) is such a machine.

Only if: Let M be an LD machine (not necessarily a $(P, S, c)'$-machine) that computes $\mathcal{Q}_c^{P,S}(W)$ and halts after a finite number of computation steps. We have to show that $W_c^{(S,P)}$ is finite. We show this by contradiction, that is, we assume $W_c^{(S,P)}$ is infinite. In this case $D_{\mathfrak{R}}$ is infinite. Since $\mathcal{Q}_c^{P,S}$ is nontrivially satisfiable it is possible that W is a Web of Linked Data for which exist solutions in $W_c^{(S,P)}$ such that each of these solutions provides a binding for at least one variable. Hence, for computing $\mathcal{Q}_c^{P,S}$ over

W completely, machine M must (recursively) expand the word on its link traversal tape until it contains the encodings of (at least) each LD document in $D_{\mathfrak{R}}$. Such an expansion is necessary to ensure that the computed query result is complete. Since $D_{\mathfrak{R}}$ is infinite the expansion requires infinitely many computing steps. However, we know that M halts after a finite number of computation steps. Hence, we have a contradiction and, thus, $W_c^{(S,P)}$ must be finite.

C.22 Proof of Proposition 8

Let c_{ef} be a reachability criterion that ensures finiteness. To prove that all SPARQL$_{\text{LD(R)}}$ queries under c_{ef}-semantics are finitely computable we have to show for each such query exists an LD machine that computes the query over any Web of Linked Data and halts after a finite number of computation steps (with an encoding of the complete query result on its output tape). W.l.o.g., let $\mathcal{Q}_{c_{ef}}^{P,S}$ be such a SPARQL$_{\text{LD(R)}}$ query (under c_{ef}-semantics). Based on Lemmas 8 to 10 (cf. Section C.21) it is easy to verify that the $(P, S, c_{ef})'$-machine (for P, S and c_{ef} as used by $\mathcal{Q}_{c_{ef}}^{P,S}$) is such a machine (notice, Lemmas 8 to 10 are not restricted to SPARQL$_{\text{LD(R)}}$ queries that are nontrivially satisfiable).

C.23 Proof of Theorem 5

Let c_{nf} be a reachability criterion that does not ensure finiteness. To prove Theorem 5 we distinguish three cases for a satisfiable SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_{c_{nf}}^{P,S}$ under c_{nf}-semantics:

1. The (S, c_{nf}, P)-reachable part of *any* Web of Linked Data is finite (which is possible even if c_{nf} does not ensure finiteness for all SPARQL$_{\text{LD(R)}}$ queries under c_{nf}-semantics).
2. The (S, c_{nf}, P)-reachable part of some Web of Linked Data is infinite and $\mathcal{Q}_{c_{nf}}^{P,S}$ is monotonic.
3. The (S, c_{nf}, P)-reachable part of some Web of Linked Data is infinite and $\mathcal{Q}_{c_{nf}}^{P,S}$ is not monotonic.

In the following we discuss each of these cases.

Case (1): Let $\mathcal{Q}_{c_{nf}}^{P',S'}$ be a satisfiable SPARQL$_{\text{LD(R)}}$ query (under c_{nf}-semantics) such that the (S', c_{nf}, P')-reachable part of *any* Web of Linked Data is finite. We claim that in this case $\mathcal{Q}_{c_{nf}}^{P',S'}$ is finitely computable (independent of its monotonicity). To prove this claim we use the same argument that we use for proving Proposition 8 in Section C.22: Based on Lemmas 8 to 10 (cf. Section C.21) and the fact that the (S', c_{nf}, P')-reachable part of *any* Web of Linked Data is finite, it is easy to verify that the $(P', S', c_{nf})'$-machine (for P', S' and c_{nf} as used by $\mathcal{Q}_{c_{nf}}^{P',S'}$) is an LD machine that computes $\mathcal{Q}_{c_{nf}}^{P',S'}$ over any Web of Linked Data W and halts after a finite number of computation steps (with an encoding of $\mathcal{Q}_{c_{nf}}^{P',S'}(W)$ on its output tape). Hence, the $(P', S', c_{nf})'$-machine satisfies the requirements in Definition 5 and, thus, $\mathcal{Q}_{c_{nf}}^{P',S'}$ is finitely computable.

Algorithm 3 The program of the (P, S, c)-machine.

1: Call `lookup` for each $u \in S$.

2: **for** $j = 1, 2, ...$ **do**

3: Let T_j denote the set of all RDF triples currently encoded on the link traversal tape. Use the work tape to enumerate the set $[\![P]\!]_{T_j}$.

4: For each $\mu \in [\![P]\!]_{T_j}$ check whether μ is already encoded on the output tape; if this is not the case, then add $\text{enc}(\mu)$ to the output.

5: Scan the link traversal tape for an RDF triple t that contains a URI $u \in \text{uris}(t)$ such that i) $c(t, u, P) = \text{true}$ and ii) the word on the link traversal tape neither contains $\text{enc}(u)\,\text{enc}(adoc(u))\,\sharp$ nor $\text{enc}(u)\,\sharp$. If such t and u exist, call `lookup` for u; otherwise halt the computation.

6: **end for**

Case (2): Let $\mathcal{Q}_{c_{nf}}^{P',S'}$ be a satisfiable, monotonic SPARQL$_{\text{LD(R)}}$ query (under c_{nf}-semantics) for which exists a Web of Linked Data W such that the (S', c_{nf}, P')-reachable part of W is infinite. To show that $\mathcal{Q}_{c_{nf}}^{P',S'}$ is (at least) eventually computable we introduce specific LD machines for SPARQL$_{\text{LD(R)}}$ queries. We call these machines (P, S, c)-machines. The (P, S, c)-machine for a SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ implements a generic (i.e. input independent) computation of $\mathcal{Q}_c^{P,S}$. We shall see that if a SPARQL$_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ is monotonic, the corresponding (P, S, c)-machine (eventually) computes $\mathcal{Q}_c^{P,S}$ over any Web of Linked Data. We emphasize that (P, S, c)-machines differ from the $(P, S, c)'$-machines that we use for proving Lemma 2 (cf. Section C.21). Formally, we define (P, S, c)-machines as follows:

Definition 19. *Let $S \subset \mathcal{U}$ be a finite set of seed URIs; let c be a reachability criterion; and let P be a SPARQL expression. The (P, S, c)-**machine** is an LD machine that implements Algorithm 3. This algorithm makes use of a subroutine called `lookup`. This subroutine, when called with a URI $u \in \mathcal{U}$, i) writes $\text{enc}(u)$ to the right end of the word on the link traversal tape, ii) enters the expand state, and iii) performs the expand procedure as specified in Definition 4.*

As can be seen in Algorithm 3, the computation of each (P, S, c)-machine (with a Web of Linked Data W encoded on its input tape) starts with an initialization (cf. line 1). After the initialization, the machine enters a (potentially non-terminating) loop. During each iteration of this loop, the machine generates valuations using all data that is currently encoded on the link traversal tape. The following proposition shows that these valuations are part of the corresponding query result (find the proof for Proposition 10 below in Section C.24):

Proposition 10. *Let:*

- *$\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\text{LD(R)}}$ query that is monotonic;*
- *$M^{(P,S,c)}$ denote the (P, S, c)-machine for P, S, and c as used by $\mathcal{Q}_c^{P,S}$; and*
- *W be an arbitrary Web of Linked Data encoded on the Web tape of $M^{(P,S,c)}$.*

During the execution of Algorithm 3 by $M^{(P,S,c)}$ it holds:

$$\forall\, j \in \{1, 2, ...\} : [\![P]\!]_{T_j} \subseteq \mathcal{Q}_c^{P,S}\big(W\big)$$

Proposition 10 presents the basis to prove the soundness of (monotonic) query results computed by Algorithm 3. To verify the completeness of these results it is important to note that (P, S, c)-machines look up no more than one URI per iteration (cf. line 5 in Algorithm 3). Hence, (P, S, c)-machines prioritize result construction over data retrieval. Due to this feature we show that for each solution in a query result exists an iteration during which that solution is computed (find the proof for Proposition 11 below in Section C.25):

Proposition 11. *Let:*

- $\mathcal{Q}_c^{P,S}$ *be a SPARQL$_{LD(R)}$ query that is monotonic;*
- $M^{(P,S,c)}$ *denote the* (P, S, c)*-machine for* P*,* S*, and* c *as used by* $\mathcal{Q}_c^{P,S}$*; and*
- W *be an arbitrary Web of Linked Data encoded on the Web tape of* $M^{(P,S,c)}$*.*

For each $\mu \in \mathcal{Q}_c^{P,S}(W)$ *exists a* $j_\mu \in \{1, 2, ...\}$ *such that during the execution of Algorithm 3 by* $M^{(P,S,c)}$ *it holds:*

$$\forall\, j \in \{j_\mu, j_\mu+1, ...\} : \mu \in [\![P]\!]_{T_j}$$

So far our results verify that i) the set of query solutions computed after any iteration is sound and ii) that this set is complete after a particular (potentially infinite) number of iterations. We now show that each iteration definitely finishes after a finite number of computation steps (find the proof for Proposition 12 below in Section C.26):

Proposition 12. *Let:*

- $M^{(P,S,c)}$ *be the* (P, S, c)*-machine for a SPARQL expression* P*, a finite set* $S \subset \mathcal{U}$*, and a reachability criterion* c*; and*
- W *be a (potentially infinite) Web of Linked Data encoded on the Web tape of* $M^{(P,S,c)}$*.*

During the execution of Algorithm 3, $M^{(P,S,c)}$ *finishes each iteration of the loop in that algorithm after a finite number of computation steps.*

Altogether, Propositions 10 to 12 conclude the discussion of case (2), that is, based on these propositions it is easy to verify that the (P', S', c_{nf})-machine for our query $\mathcal{Q}_{c_{nf}}^{P',S'}$ satisfies the requirements in Definition 6 and, thus, $\mathcal{Q}_{c_{nf}}^{P',S'}$ is eventually computable.

Case (3): Let $\mathcal{Q}_{c_{nf}}^{P',S'}$ be a satisfiable, non-monotonic SPARQL$_{\mathsf{LD(R)}}$ query (under c_{nf}-semantics) for which exists a Web of Linked Data W such that the (S', c_{nf}, P')-reachable part of W is infinite. To show that $\mathcal{Q}_{c_{nf}}^{P',S'}$ may not even be eventually computable, we assume $\mathcal{Q}_{c_{nf}}^{P',S'}(W) \neq \varnothing$.

For the prove we use the same argument that we use in the corresponding discussion for non-monotonic SPARQL$_{\mathsf{LD}}$ queries (see the proof of Theorem 1 in Section C.5). Hence, we show a contradiction by assuming $\mathcal{Q}_{c_{nf}}^{P',S'}$ were (at least) eventually computable, that is, we assume an LD machine M (which is not necessarily a (P, S, c)-machine) whose computation of $\mathcal{Q}_{c_{nf}}^{P',S'}$ on any Web of Linked Data has the two properties given in Definition 6. To obtain a contradiction we show that such a machine does not exist.

Let W be a Web of Linked Data such that the (S', c_{nf}, P')-reachable part of W is infinite; such a Web exists for case (3). In the remainder of this proof we write $W_{\mathfrak{R}}$ to denote the (S', c_{nf}, P')-reachable part of W.

Let W be encoded on the Web tape of M and let μ be an arbitrary solution for $\mathcal{Q}_{c_{nf}}^{P',S'}$ in W; i.e. $\mu \in \mathcal{Q}_{c_{nf}}^{P',S'}(W)$. Based on our assumption, machine M must write $\mathrm{enc}(\mu)$ to its output tape after a finite number of computation steps (cf. property 2 in Definition 6). We argue that this is impossible: Since $\mathcal{Q}_{c_{nf}}^{P',S'}$ is not monotonic, M cannot add μ to the output before M has accessed all LD documents in $W_{\mathfrak{R}}$ (i.e. all LD documents that are (c_{nf}, P')-reachable from S' in W). However, due to the infiniteness of $W_{\mathfrak{R}}$, there is an infinite number of such documents. Therefore, accessing all these documents is a non-terminating process and, thus, M cannot write μ to its output after a finite number of computation steps. As a consequence, the computation of $\mathcal{Q}_{c_{nf}}^{P',S'}$ (over W) by M does not have the properties given in Definition 6, which contradicts our initial assumption. Due to this contradiction we may conclude that $\mathcal{Q}_{c_{nf}}^{P',S'}$ is not eventually computable.

C.24 Proof of Proposition 10

Let:

- $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\text{LD(R)}}$ query that is monotonic;
- $M^{(P,S,c)}$ denote the (P, S, c)-machine for P, S, and c as used by $\mathcal{Q}_c^{P,S}$; and
- $W = (D, data, adoc)$ be an arbitrary Web of Linked Data encoded on the Web tape of $M^{(P,S,c)}$.

To prove Proposition 10 we use the following lemma.

Lemma 11. *During the execution of Algorithm 3 by $M^{(P,S,c)}$ on (Web) input $\mathrm{enc}(W)$ it holds $\forall\, j \in \{1, 2, ...\} : T_j \subseteq \mathrm{AllData}\big(W_c^{(S,P)}\big)$.*

Proof of Lemma 11. This proof resembles the proof of the corresponding lemma for $M^{(P,S,c)'}$ machines (cf. Lemma 9 in Section C.21). Let w_j be the word on the link traversal tape of $M^{(P,S,c)}$ when $M^{(P,S,c)}$ starts the j-th iteration of the main processing loop in Algorithm 3 (i.e. before line 3).

To prove $\forall\, j \in \{1, 2, ...\} : T_j \subseteq \mathrm{AllData}\big(W_c^{(S,P)}\big)$ it is sufficient to show for each w_j (where $j \in \{1, 2, ...\}$) exists a finite sequence $u_1, ..., u_{n_j}$ of n_j different URIs $u_i \in \mathcal{U}$ (where $i \in [1, n_j]$) such that i) w_j is[8]

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,...\,\sharp\,\mathrm{enc}(u_{n_j})\,\mathrm{enc}(adoc(u_{n_j}))\,\sharp$$

and ii) for each $i \in [1, n_j]$ either $u_i \notin \mathrm{dom}(adoc)$ (and, thus, $adoc(u_i)$ is undefined) or $adoc(u_i)$ is an LD document which is (c, P)-reachable from S in W. We use an induction over j for this proof.

Base case ($j = 1$): The computation of $M^{(P,S,c)}$ starts with an empty link traversal tape (cf. Definition 4). Due to the initialization, w_1 is a concatenation of sub-words

[8] We, again, assume $\mathrm{enc}(adoc(u_i))$ is the empty word if $u_i \notin \mathrm{dom}(adoc)$.

$\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ for all $u \in S$ (cf. line 1 in Algorithm 3). Hence, we have a corresponding sequence $u_1, \ldots, u_{n_1}$ where $n_1 = |S|$ and $\forall\, i \in [1, n_1] : u_i \in S$. The order of the URIs in that sequence depends on the order in which they have been looked up and is irrelevant for our proof. For all $u \in S$ it holds either $u \notin \mathrm{dom}(adoc)$ or $adoc(u)$ is (c, P)-reachable from S in W (cf. case 1 in Definition 10).

Induction step ($j > 1$): Our inductive hypothesis is that there exists a finite sequence $u_1, \ldots, u_{n_{j-1}}$ of n_{j-1} different URIs ($\forall\, i \in [1, n_{j-1}] : u_i \in \mathcal{U}$) such that i) w_{j-1} is

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp \ldots \sharp\,\mathrm{enc}(u_{n_{j-1}})\,\mathrm{enc}(adoc(u_{n_{j-1}}))\,\sharp$$

and ii) for each $i \in [1, n_{j-1}]$ either $u_i \notin \mathrm{dom}(adoc)$ or $adoc(u_i)$ is (c, P)-reachable from S in W. In the (j-1)-th iteration $M^{(P,S,c)}$ finds an RDF triple t encoded as part of w_{j-1} such that $\exists\, u \in \mathrm{uris}(t) : c(t, u, P) = \mathrm{true}$ and `lookup` has not been called for u. The machine calls `lookup` for u, which changes the word on the link traversal tape to w_j. Hence, w_j is equal to $w_{j-1}\,\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ and, thus, our sequence of URIs for w_j is $u_1, \ldots, u_{n_{j-1}}, u$. It remains to show that if $u \in \mathrm{dom}(adoc)$ then $adoc(u)$ is (c, P)-reachable from S in W.

Assume $u \in \mathrm{dom}(adoc)$. Since RDF triple t is encoded as part of w_{j-1} we know, from our inductive hypothesis, that t must be contained in the data of an LD document d^* that is (c, P)-reachable from S in W (and for which exists $i \in [1, n_{j-1}]$ such that $adoc(u_i) = d^*$). Therefore, t and u satisfy the requirements as given in case 2 of Definition 10 and, thus, $adoc(u)$ is (c, P)-reachable from S in W. □

Due to the monotonicity of $\mathcal{Q}_c^{P,S}$ it is trivial to show Proposition 10 using Lemma 11 (recall, $\mathcal{Q}_c^{P,S}\big(W\big) = [\![P]\!]_{\mathrm{AllData}\left(W_c^{(S,P)}\right)}$).

C.25 Proof of Proposition 11

Let:

- $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that is monotonic;
- $M^{(P,S,c)}$ denote the (P, S, c)-machine for P, S, and c as used by $\mathcal{Q}_c^{P,S}$; and
- $W = (D, data, adoc)$ be an arbitrary Web of Linked Data encoded on the Web tape of $M^{(P,S,c)}$.

To prove Proposition 11 we use the following lemma.

Lemma 12. *For each RDF triple $t \in \mathrm{AllData}\big(W_c^{(S,P)}\big)$ exists a $j_t \in \{1, 2, \ldots\}$ such that during the execution of Algorithm 3 by $M^{(P,S,c)}$ on (Web) input $\mathrm{enc}(W)$ it holds $\forall\, j \in \{j_t, j_t+1, \ldots\} : t \in T_j$.*

Proof of Lemma 12. Let w_j be the word on the link traversal tape of $M^{(P,S,c)}$ when $M^{(P,S,c)}$ starts the j-th iteration of the main processing loop in Algorithm 3 (i.e. before line 3).

W.l.o.g., let t' be an arbitrary RDF triple $t' \in \mathrm{AllData}\big(W_c^{(S,P)}\big)$. There must exist an LD document $d \in D$ such that i) $t' \in data(d)$ and ii) d is (c, P)-reachable from S

in W. Let d' be such a document. Since $M^{(P,S,c)}$ only appends to its link traversal tape we prove that there exists a $j_{t'} \in \{1, 2, ...\}$ with $\forall\, j \in \{j_{t'}, j_{t'}+1, ...\} : t' \in T_j$ by showing that there exists $j_{t'} \in \{1, 2, ...\}$ such that $w_{j_{t'}}$ contains the sub-word $\mathrm{enc}(d')$. This proof resembles the proof of the corresponding lemma for $M^{(P,S,c)'}$ machines (cf. Lemma 8 in Section C.21).

Since d' is (c, P)-reachable from S in W, the link graph for W contains at least one finite path $(d_0, ..., d_n)$ of LD documents d_i where i) $n \in \{0, 1, ...\}$, ii) $d_n = d'$, iii) $\exists\, u \in S : adoc(u) = d_0$, and iv) for each $i \in \{1, ..., n\}$ it holds:

$$\exists\, t \in data(d_{i-1}) : \Big(\exists\, u \in \mathrm{uris}(t) : \big(adoc(u) = d_i \text{ and } c(t, u, P) = \mathrm{true}\big)\Big) \qquad (2)$$

Let $(d_0^*, ..., d_n^*)$ be such a path. We use this path for our proof. More precisely, we show by induction over $i \in \{0, ..., n\}$ that there exists $j_t \in \{1, 2, ...\}$ such that w_{j_t} contains the sub-word $\mathrm{enc}(d_n^*)$ (which is the same as $\mathrm{enc}(d')$ because $d_n^* = d'$).

Base case ($i = 0$): Since $\exists\, u \in S : adoc(u) = d_0^*$ it is easy to verify that w_1 contains the sub-word $\mathrm{enc}(d_0^*)$ (cf. line 1 in Algorithm 3).

Induction step ($i > 0$): Our inductive hypothesis is: There exists $j \in \{1, 2, ...\}$ such that w_j contains sub-word $\mathrm{enc}(d_{i-1}^*)$. Based on the hypothesis we show that there exists a $j' \in \{j, j+1, ...\}$ such that $w_{j'}$ contains the sub-word $\mathrm{enc}(d_i^*)$. We distinguish two cases: either $\mathrm{enc}(d_i^*)$ is already contained in w_j or it is not contained in w_j. In the first case we have $j' = j$; in the latter case we have $j' > j$. We have to discuss the latter case only. Due to (2) exist $t^* \in data(d_{i-1}^*)$ and $u^* \in \mathrm{uris}(t^*)$ such that $adoc(u^*) = d_i^*$ and $c(t^*, u^*, P) = \mathrm{true}$. Hence, there exists a $\delta \in \mathbb{N}^0$ such that $M^{(P,S,c)}$ finds t^* and u^* in the (j+δ)-th iteration. Since $M^{(P,S,c)}$ calls `lookup` for u^* in that iteration (cf. line 5 in Algorithm 3), it holds that $w_{j+\delta+1}$ contains $\mathrm{enc}(d_i^*)$ and, thus, $j' = j + \delta + 1$. □

We now prove Proposition 11 by induction over the structure of possible SPARQL expressions. This proof resembles the proof of Lemma 4 (cf. Section C.7).

Base case: Assume that SPARQL expression P is a triple pattern tp. W.l.o.g., let $\mu \in \mathcal{Q}_c^{P,S}(W)$. It holds $\mathrm{dom}(\mu) = \mathrm{vars}(tp)$ and $t = \mu[tp] \in \mathrm{AllData}\big(W_c^{(S,P)}\big)$ (cf. Definitions 12 and 15). According to Lemma 12 exists a $j_\mu \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_\mu, j_\mu + 1, ...\} : t \in T_j$. Since $\mathcal{Q}_c^{P,S}$ is monotonic we conclude $\forall\, j \in \{j_\mu, j_\mu+1, ...\} : \mu \in [\![P]\!]_{T_j}$.

Induction step: Our inductive hypothesis is that for SPARQL expressions P_1 and P_2 it holds:

1. For each $\mu \in \mathcal{Q}_c^{P_1,S}(W)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 3 by $M^{(P,S,c)}$ it holds $\forall\, j \in \{j_\mu, j_\mu+1, ...\} : \mu \in [\![P_1]\!]_{T_j}$; and
2. For each $\mu \in \mathcal{Q}_c^{P_2,S}(W)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 3 by $M^{(P,S,c)}$ it holds $\forall\, j \in \{j_\mu, j_\mu+1, ...\} : \mu \in [\![P_2]\!]_{T_j}$.

Based on this hypothesis we show that for any SPARQL expression P that can be constructed using P_1 and P_2 it holds: For each $\mu \in \mathcal{Q}_c^{P,S}(W)$ exists a $j_\mu \in \{1, 2, ...\}$ such that during the execution of Algorithm 3 by $M^{(P,S,c)}$ it holds $\forall\, j \in \{j_\mu, j_\mu+1, ...\} :$

$\mu \in [\![P]\!]_{T_j}$. W.l.o.g., let $\mu' \in \mathcal{Q}_c^{P,S}(W)$. According to Definition 14 we distinguish the following cases:

- P is $(P_1$ AND $P_2)$. In this case exist $\mu_1 \in \mathcal{Q}_c^{P_1,S}(W)$ and $\mu_2 \in \mathcal{Q}_c^{P_2,S}(W)$ such that $\mu' = \mu_1 \cup \mu_2$ and $\mu_1 \sim \mu_2$. According to our inductive hypothesis exist $j_{\mu_1}, j_{\mu_2} \in \{1, 2, ...\}$ such that i) $\forall\, j \in \{j_{\mu_1}, j_{\mu_1}+1, ...\} : \mu_1 \in [\![P_1]\!]_{T_j}$ and ii) $\forall\, j \in \{j_{\mu_2}, j_{\mu_2}+1, ...\} : \mu_2 \in [\![P_2]\!]_{T_j}$. Let $j_{\mu'} = \max(\{j_{\mu_1}, j_{\mu_2}\})$. Due to the monotonicity of $\mathcal{Q}_c^{P,S}$ it holds $\forall\, j \in \{j_{\mu'}, j_{\mu'}+1, ...\} : \mu' \in [\![P]\!]_{T_j}$.
- P is $(P_1$ FILTER $R)$. In this case exist $\mu^* \in \mathcal{Q}_c^{P_1,S}(W)$ such that $\mu' = \mu^*$. According to our inductive hypothesis exist $j_{\mu^*} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}+1, ...\} : \mu^* \in [\![P_1]\!]_{T_j}$. Due to the monotonicity of $\mathcal{Q}_c^{P,S}$ it holds $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}+1, ...\} : \mu' \in [\![P]\!]_{T_j}$.
- P is $(P_1$ OPT $P_2)$. We distinguish two cases:
  1. There exist $\mu_1 \in \mathcal{Q}_c^{P_1,S}(W)$ and $\mu_2 \in \mathcal{Q}_c^{P_2,S}(W)$ such that $\mu' = \mu_1 \cup \mu_2$ and $\mu_1 \sim \mu_2$. This case corresponds to the case where P is $(P_1$ AND $P_2)$ (see above).
  2. There exist $\mu_1 \in \mathcal{Q}_c^{P_1,S}(W)$ such that $\mu' = \mu_1$ and $\forall\, \mu_2 \in \mathcal{Q}_c^{P_2,S}(W) : \mu_1 \not\sim \mu_2$. According to our inductive hypothesis exist $j_{\mu_1} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu_1}, j_{\mu_1}+1, ...\} : \mu_1 \in [\![P_1]\!]_{T_j}$. Due to the monotonicity of $\mathcal{Q}_c^{P,S}$ it holds $\forall\, j \in \{j_{\mu_1}, j_{\mu_1}+1, ...\} : \mu' \in [\![P]\!]_{T_j}$.
- P is $(P_1$ UNION $P_2)$. We distinguish two cases:
  1. There exists $\mu^* \in \mathcal{Q}_c^{P_1,S}(W)$ such that $\mu' = \mu^*$. According to our inductive hypothesis exist $j_{\mu^*} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}+1, ...\} : \mu^* \in [\![P_1]\!]_{T_j}$.
  2. There exists $\mu^* \in \mathcal{Q}_c^{P_2,S}(W)$ such that $\mu' = \mu^*$. According to our inductive hypothesis exist $j_{\mu^*} \in \{1, 2, ...\}$ such that $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}+1, ...\} : \mu^* \in [\![P_2]\!]_{T_j}$.

  Due to the monotonicity of $\mathcal{Q}_c^{P,S}$ it holds for both cases: $\forall\, j \in \{j_{\mu^*}, j_{\mu^*}+1, ...\} : \mu' \in [\![P]\!]_{T_j}$.

C.26 Proof of Proposition 12

Let:

- $M^{(P,S,c)}$ be the (P, S, c)-machine for a SPARQL expression P, a finite set $S \subset \mathcal{U}$, and a reachability criterion c; and
- W be a (potentially infinite) Web of Linked Data encoded on the Web tape of $M^{(P,S,c)}$.

To prove that $M^{(P,S,c)}$ finishes each iteration of the loop in Algorithm 3 after a finite number of computation steps, we first emphasize the following facts:

1. Each call of subroutine `lookup` by $M^{(P,S,c)}$ terminates because the encoding of W is ordered following the order of the URIs in $\mathrm{dom}(adoc)$.
2. $M^{(P,S,c)}$ completes the initialization in line 1 of Algorithm 3 after a finite number of steps because S is finite.

3. At any point in the computation the word on the link traversal tape of $M^{(P,S,c)}$ is finite because $M^{(P,S,c)}$ only gradually appends (encoded) LD documents to that tape (one document per iteration) and the encoding of each document is finite (recall that the set of RDF triples $data(d)$ for each LD document d is finite).

It remains to show that each iteration of the loop also only requires a finite number of computation steps: Due to the finiteness of the word on the link traversal tape, each $[\![P]\!]_{T_j}$ (for $j = 1, 2, ...$) is finite, resulting in a finite number of computation steps for lines 3 and 4 during any iteration. The scan in line 5 also finishes after a finite number of computation steps because of the finiteness of the word on the link traversal tape.

C.27 Proof of Theorem 6

We formally define the termination problem for $\text{SPARQL}_{\text{LD(R)}}$ as follows:

Problem:	TERMINATION($\text{SPARQL}_{\text{LD(R)}}$)
Web Input:	a (potentially infinite) Web of Linked Data W
Ordinary Input:	a finite but nonempty set $S \subset \mathcal{U}$ of seed URIs
	a reachability criterion c_{nf} that does not ensure finiteness
	a SPARQL expression P
Question:	Does an LD machine exist that computes $\mathcal{Q}^{P,S}_{c_{nf}}(W)$ and halts?

To prove that TERMINATION($\text{SPARQL}_{\text{LD(R)}}$) is not LD machine decidable we reduce the halting problem to TERMINATION($\text{SPARQL}_{\text{LD(R)}}$). For this reduction we use the same argumentation, including the same Web of Linked Data, that we use for proving Theorem 2 (cf. Section C.9).

We define the mapping from input for the halting problem to input for TERMINATION($\text{SPARQL}_{\text{LD(R)}}$) as follows: Let (w, x) be an input to the halting problem, that is, w is the description of a Turing machine $M(w)$ and x is a possible input word for $M(w)$); then $f(w, x) = \big(W_{\mathsf{TMs}}, S_{w,x}, c_{\mathsf{All}}, P_{w,x}\big)$ where:

- W_{TMs} is the Web of Linked Data defined in Section C.9,
- $S_{w,x} = \big\{u_1^{w,x}\big\}$ (recall, $u_1^{w,x}$ denotes a URI that identifies the first step in the computation of $M(w)$ on input x), and
- $P_{w,x} = (u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$.

As before, f is computable by Turing machines (including LD machines).

To show that TERMINATION($\text{SPARQL}_{\text{LD(R)}}$) is not LD machine decidable, suppose it were LD machine decidable. In such a case an LD machine could answer the halting problem for any input (w, x) as follows: $M(w)$ halts on x if and only if an LD machine exists that computes $\mathcal{Q}^{P,S}_{c_{nf}}\big(W_{\mathsf{TMs}}\big)$ and halts. However, we know the halting problem is undecidable for TMs (which includes LD machines). Hence, we have a contradiction and, thus, TERMINATION($\text{SPARQL}_{\text{LD(R)}}$) cannot be LD machine decidable.

C.28 Proof of Proposition 9

Let:

– $\mathcal{Q}^{P,S}_{c_{nf}}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses a finite, nonempty set $S \subset \mathcal{U}$ of seed URIs and a reachability criterion c_{nf} which does not ensure finiteness; and
– G_1, G_2 be an arbitrary pair of set of RDF triples such that $G_1 \subseteq G_2$.

Assume $\mathcal{Q}^{P,S}_{c_{nf}}$ is monotonic. We have to show that the SPARQL expression P (used by $\mathcal{Q}^{P,S}_{c_{nf}}$) is monotonic as well. We distinguish two cases: either P is satisfiable or P is not satisfiable. In the latter case P is trivially monotonic. Hence, we only have to discuss the first case. To prove that (the satisfiable) P is monotonic it suffices to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. For this proof we construct two Webs of Linked Data W_1 and W_2 such that i) W_1 is an induced subweb of W_2 and ii) the data of G_1 and G_2 is distributed over W_1 and W_2, respectively. Using W_1 and W_2 we show the monotonicity of P based on the monotonicity of $\mathcal{Q}^{P,S}_{c_{nf}}$.

To construct W_1 and W_2 we have to address two problems: First, we cannot simply construct W_1 and W_2 as Webs of Linked Data that consist of single LD documents which contain all RDF triples of G_1 and G_2 because G_1 and G_2 may be (countably) infinite, whereas the data in each LD document of a Web of Linked Data must be finite. Recall the corresponding proof for SPARQL$_{\mathsf{LD}}$ where we have the same problem (cf. Section C.4); we shall use the same strategy for solving that problem in this proof. The second problem, however, is specific to the case of reachability-based semantics: The construction of W_1 and W_2 for SPARQL$_{\mathsf{LD(R)}}$ queries has to ensure that all LD documents which contain RDF triples of G_1 and G_2 are reachable. Due to this issue the construction is more complex than the corresponding construction for the full-Web semantics case.

To solve the first problem we construct W_1 and W_2 as Webs that contain an LD document for each RDF triples in G_1 and G_2, respectively. However, by distributing the RDF triples from (the potentially infinite) G_1 over multiple LD documents in a constructed Web, we may lose certain solutions $\mu \in [\![P]\!]_{G_1}$ because the data of each LD document in a Web of Linked Data must use a unique set of blank nodes. The same holds for G_2. To avoid this issue we assume a mapping ϱ that maps each blank node in G_2 to a new, unique URI. To define ϱ formally, we let B denote the set of blank nodes in G_2, that is, $B = \mathrm{terms}(G_2) \cap \mathcal{B}$. Furthermore, we assume a set $U_B \subset \mathcal{U}$ such that $|U_B| = |B|$ and $U_B \cap \mathrm{terms}(G_2) = \varnothing$. Now, ϱ is a total, bijective mapping $\varrho : \big((\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \setminus U_B\big) \to \big((\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \setminus B\big)$ that, for any $x \in \big((\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \setminus U_B\big)$, is defined as follows:

$$\varrho(x) = \begin{cases} \varrho_B(x) & \text{if } x \in B, \\ x & \text{else.} \end{cases}$$

where ϱ_B is an arbitrary bijection $\varrho_B : B \to U_B$.

The application of ϱ to an arbitrary RDF triple $t = (x_1, x_2, x_3)$, denoted by $\varrho[t]$, results in an RDF triple $t' = (x'_1, x'_2, x'_3)$ such that $x'_i = \varrho(x_i)$ for all $i \in \{1, 2, 3\}$. Furthermore, the application of ϱ to a valuation μ, denoted by $\varrho[\mu]$, results in a valuation μ' such that $\mathrm{dom}(\mu') = \mathrm{dom}(\mu)$ and $\mu'(?v) = \varrho(\mu(?v))$ for all $?v \in \mathrm{dom}(\mu)$.

We now let

$$G'_1 = \big\{\varrho[t] \,\big|\, t \in G_1\big\} \qquad \text{and} \qquad G'_2 = \big\{\varrho[t] \,\big|\, t \in G_2\big\}$$

The following facts are verified easily:

Fact 4. *It holds:* $G'_1 \subseteq G'_2$, $|G_1| = |G'_1|$, *and* $|G_2| = |G'_2|$.

Fact 5. *For all* $j \in \{1, 2\}$ *it holds: Let* μ *be an arbitrary valuation, then* $\varrho[\mu]$ *is a solution for* P *in* G'_j *if and only if* μ *is a solution for* P *in* G_j. *More precisely:*

$$\forall\, \mu \in [\![P]\!]_{G_j} : \varrho[\mu] \in [\![P]\!]_{G'_j} \qquad \textit{and} \qquad \forall\, \mu' \in [\![P]\!]_{G'_j} : \varrho^{-1}[\mu'] \in [\![P]\!]_{G_j}$$

where ϱ^{-1} *denotes the inverse of the bijective mapping* ϱ.

We now address the second problem, that is, we construct W_1 and W_2 (using G'_1 and G'_2) in a way that all LD documents which contain RDF triples from G'_1 and G'_2 are reachable. To achieve this goal we use a reachable part of another Web of Linked Data for the construction. We emphasize that this reachable part must be infinite because G_1 and G_2 may be (countably) infinite. To find a Web of Linked Data with such a reachable part we make use of c_{nf}: Since c_{nf} does not ensure finiteness, we know there exists a Web of Linked Data $W^* = (D^*, data^*, adoc^*)$, a (finite, nonempty) set $S^* \subset \mathcal{U}$ of seed URIs, and a SPARQL expression P^* such that the (S^*, c_{nf}, P^*)-reachable part of W^* is infinite. Notice, S^* and P^* are not necessarily the same as S and P.

While the (S^*, c_{nf}, P^*)-reachable part of W^* presents the basis for our proof, we cannot use it directly because the data in that part may cause undesired side-effects for the evaluation of P. To avoid this issue we define an isomorphism σ for W^*, S^*, and P^* such that the images of W^*, S^*, and P^* under σ do not use any RDF term or query variable from G'_2 and P.

For the definition of σ we write U, L, and V to denote the sets of all URIs, literals, and variables in G'_2 and P (recall, neither G'_2 nor P contain blank nodes). That is:

$$\begin{aligned} U &= \big(\mathrm{terms}(G'_2) \cup \mathrm{terms}(P)\big) \cap \mathcal{U}, \\ L &= \big(\mathrm{terms}(G'_2) \cup \mathrm{terms}(P)\big) \cap \mathcal{L}, \text{ and} \\ V &= \mathrm{vars}(P) \cup \mathrm{vars}_{\mathsf{F}}(P) \end{aligned}$$

where $\mathrm{vars}_{\mathsf{F}}(P)$ denotes the set of all variables in all filter conditions of P (if any). Similarly to U, L, and V, we write U^*, L^*, and V^* to denote the sets of all URIs, literals, and variables in W^*, S^*, and P^*:

$$\begin{aligned} U^* &= S^* \cup \mathrm{terms}\big(\mathrm{AllData}(W^*)\big) \cap \mathcal{U}, \\ L^* &= \mathrm{terms}\big(\mathrm{AllData}(W^*)\big) \cap \mathcal{L}, \quad \text{and} \\ V^* &= \mathrm{vars}(P^*) \cup \mathrm{vars}_{\mathsf{F}}(P^*). \end{aligned}$$

Moreover, we assume three new sets of URIs, literals, and variables, denoted by U_{new}, L_{new}, and V_{new}, respectively. For these sets it must hold:

$$\begin{aligned} &U_{\mathsf{new}} \subset \mathcal{U} \text{ such that } |U_{\mathsf{new}}| = |U| \text{ and } U_{\mathsf{new}} \cap (U \cup U^*) = \varnothing; \\ &L_{\mathsf{new}} \subset \mathcal{L} \text{ such that } |L_{\mathsf{new}}| = |L| \text{ and } L_{\mathsf{new}} \cap (L \cup L^*) = \varnothing; \text{ and} \\ &V_{\mathsf{new}} \subset \mathcal{V} \text{ such that } |V_{\mathsf{new}}| = |V| \text{ and } V_{\mathsf{new}} \cap (V \cup V^*) = \varnothing. \end{aligned}$$

Furthermore, we assume three total, bijective mappings:

$$\sigma_U : U \to U_{\mathsf{new}} \qquad \sigma_L : L \to L_{\mathsf{new}} \qquad \sigma_V : V \to V_{\mathsf{new}}$$

Now we define σ as a total, bijective mapping

$$\sigma : \Big(\big(\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}\big) \setminus \big(U_{\mathsf{new}} \cup L_{\mathsf{new}} \cup V_{\mathsf{new}}\big) \Big) \to \Big(\big(\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}\big) \setminus \big(U \cup L \cup V\big) \Big)$$

such that for each $x \in \mathrm{dom}(\sigma)$ it holds:

$$\sigma(x) = \begin{cases} \sigma_U(x) & \text{if } x \in U, \\ \sigma_L(x) & \text{if } x \in L, \\ \sigma_V(x) & \text{if } x \in V, \\ x & \text{else.} \end{cases}$$

The application of σ to an arbitrary valuation μ and to an arbitrary RDF triple is defined in a way that corresponds to the application of ϱ to μ and t, respectively. An application of σ to further, relevant structures is defined as follows:

- The application of σ to the aforementioned Web $W^* = (D^*, data^*, adoc^*)$, denoted by $\sigma[W^*]$, results in a Web of Linked Data $W^{*\prime} = (D^{*\prime}, data^{*\prime}, adoc^{*\prime})$ such that $D^{*\prime} = D^*$ and mappings $data^{*\prime}$ and $adoc^{*\prime}$ are defined as follows:

$$\forall\, d \in D^{*\prime} : data^{*\prime}(d) = \big\{ \sigma[t] \,\big|\, t \in data^*(d) \big\}$$
$$\forall\, u \in \mathrm{dom}(adoc^{*\prime}) : adoc^{*\prime}(u) = adoc^*\big(\sigma^{-1}(u)\big)$$

  where $\mathrm{dom}(adoc^{*\prime}) = \big\{ \sigma(u) \,\big|\, u \in \mathrm{dom}(adoc^*) \big\}$ and σ^{-1} is the inverse of σ.
- The application of σ to an arbitrary (SPARQL) filter condition R, denoted by $\sigma[R]$, results in a filter condition that is defined as follows: i) If R is $?x = c$, $?x = ?y$, or $\mathrm{bound}(?x)$, then $\sigma[R]$ is $?x' = c'$, $?x' = ?y'$, and $\mathrm{bound}(?x')$, respectively, where $?x' = \sigma(?x)$, $?y' = \sigma(?y)$, and $c' = \sigma(c)$; and ii) If R is $(\neg R_1)$, $(R_1 \wedge R_2)$, or, $(R_1 \vee R_2)$, then $\sigma[R]$ is $(\neg R_1')$, $(R_1' \wedge R_2')$, or, $(R_1' \vee R_2')$, respectively, where $R_1' = \sigma[R_1]$ and $R_2' = \sigma[R_2]$.
- The application of σ to an arbitrary SPARQL expression P', denoted by $\sigma[P']$, results in a SPARQL expression that is defined as follows: i) If P' is a triple pattern $\big(x_1', x_2', x_3'\big)$, then $\sigma[P']$ is (x_1'', x_2'', x_3'') such that $x_i'' = \sigma(x_i')$ for all $i \in \{1, 2, 3\}$; and ii) If P' is $(P_1' \ \mathsf{AND}\ P_2')$, $(P_1' \ \mathsf{UNION}\ P_2')$, $(P_1' \ \mathsf{OPT}\ P_2')$, or $(P_1' \ \mathsf{FILTER}\ R')$, then $\sigma[P']$ is $(P_1'' \ \mathsf{AND}\ P_2'')$, $(P_1'' \ \mathsf{UNION}\ P_2'')$, or $(P_1'' \ \mathsf{OPT}\ P_2'')$, and $(P_1'' \ \mathsf{FILTER}\ R'')$, respectively, where $P_1'' = \sigma[P_1']$, $P_2'' = \sigma[P_2']$, and $R'' = \sigma[R']$.

We now introduce $W^{*\prime}$, $S^{*\prime}$, and $P^{*\prime}$ as image of W^*, S^*, and P^* under σ, respectively:

$$W^{*\prime} = \sigma[W^*] \qquad S^{*\prime} = \big\{ \sigma(u) \,\big|\, u \in S^* \big\} \qquad P^{*\prime} = \sigma[P^*]$$

$W^{*\prime}$ is structurally identical to W^*. Furthermore, the $(S^{*\prime}, c_{nf}, P^{*\prime})$-reachable part of $W^{*\prime}$ is infinite because the (S^*, c_{nf}, P^*)-reachable part of W^* is infinite. Hereafter, we write $W_{\mathfrak{R}} = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ to denote the $(S^{*\prime}, c_{nf}, P^{*\prime})$-reachable part of $W^{*\prime}$.

We now use $W_{\mathfrak{R}}$ to construct Webs of Linked Data that contain all RDF triples from G_1' and G_2', respectively. Since $W_{\mathfrak{R}}$ is infinite, there exists at least one infinite path in

the link graph of $W_{\mathfrak{R}}$. Let $p = d_1, d_2, ...$ be such a path. Hence, for all $i \in \{1, 2, ...\}$ holds:

$$d_i \in D_{\mathfrak{R}} \qquad \text{and} \qquad \exists\, t \in data_{\mathfrak{R}}(d_i) : \Big(\exists\, u \in \text{uris}(t) : adoc_{\mathfrak{R}}(u) = d_{i+1}\Big)$$

We may use this path for constructing Webs of Linked Data W_1 and W_2 from $W_{\mathfrak{R}}$ such that W_1 and W_2 contain the data from G'_1 and G'_2, respectively. However, to allow us to use the monotonicity of SPARQL$_{\text{LD(R)}}$ queries in our proof, it is necessary to construct W_1 and W_2 such that W_1 is an induced subweb of W_2. To achieve this goal we assume a strict total order on G'_2 such that each $t \in G'_1 \subseteq G'_2$ comes before any $t' \in G'_2 \setminus G'_1$ in that order. Formally, we denote this order by infix $<$ and, thus, require $\forall\,(t, t') \in G'_1 \times (G'_2 \setminus G'_1) : t < t'$. Furthermore, we assume a total, injective function $pdoc : G'_2 \to \big\{d \in D_{\mathfrak{R}} \,\big|\, d \text{ is on path } p\big\}$ which is order-preserving, that is, for each pair $(t, t') \in G'_2 \times G'_2$ holds: If $t < t'$ then LD document $pdoc(t)$ comes before LD document $pdoc(t')$ on path p.

We now use $pdoc$, G'_2, and $W_{\mathfrak{R}} = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ to construct a Web of Linked Data $W_2 = (D_2, data_2, adoc_2)$ as follows:

$$D_2 = D_{\mathfrak{R}}$$

$$\forall\, d \in D_2 : data_2(d) = \begin{cases} data_{\mathfrak{R}}(d) \cup \{t\} & \text{if } \exists\, t \in G'_2 : pdoc(t) = d, \\ data_{\mathfrak{R}}(d) & \text{else.} \end{cases}$$

$$\forall\, u \in \text{dom}(adoc_{\mathfrak{R}}) : adoc_2(u) = adoc_{\mathfrak{R}}(u)$$

In addition to W_2, we introduce a Web of Linked Data $W_1 = (D_1, data_1, adoc_1)$ that is an induced subweb of W_2 and that is defined by[9]:

$$D_1 = \big\{d \in D_2 \,\big|\, \text{either } d \text{ is not on path } p \text{ or } \exists\, t \in G'_1 : d = pdoc(t)\big\}$$

The following facts are verified easily:

Fact 6. *For all $j \in \{1, 2\}$ it holds: $G'_j \subset \text{AllData}\big(W_j\big) = G'_j \cup \text{AllData}\big(W_{\mathfrak{R}}\big)$.*

Fact 7. *For all $j \in \{1, 2\}$ it holds: The $(S^{*\prime}, c_{nf}, P^{*\prime})$-reachable part of W_j is W_j itself.*

Fact 8. *For all $j \in \{1, 2\}$ it holds: $[\![P]\!]_{G'_j} = [\![P]\!]_{\text{AllData}(W_j)}$.*

We now consider a SPARQL expression $(P \text{ UNION } P^{*\prime})$. In the following we write $\tilde{P}$ to denote this expression. Since $\text{terms}\big(G'_2\big) \cap \text{terms}\big(\text{AllData}(W_{\mathfrak{R}})\big) = \varnothing$ we conclude the following facts:

Fact 9. *For all $j \in \{1, 2\}$ it holds:*

1. *The $(S^{*\prime}, c_{nf}, \tilde{P})$-reachable part of W_j is W_j itself.*
2. $[\![P]\!]_{\text{AllData}(W_j)} \cup [\![P^{*\prime}]\!]_{\text{AllData}(W_j)} = [\![\tilde{P}]\!]_{\text{AllData}(W_j)}$
3. $[\![P]\!]_{\text{AllData}(W_j)} \cap [\![P^{*\prime}]\!]_{\text{AllData}(W_j)} = \varnothing$

[9] Recall, any induced subweb is unambiguously defined by specifying its set of LD documents.

Since W_1 is an induced subweb of W_2, $\mathcal{Q}^{P,S}_{c_{nf}}$ is monotonic, and $\tilde{P}$ is $(P \text{ UNION } P^{*\prime})$, we conclude the following inclusion from Fact 9 and Definition 12:

$$\big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_1) \setminus [\![P^{*\prime}]\!]_{\mathrm{AllData}(W_1)}\big) \subseteq \big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_2) \setminus [\![P^{*\prime}]\!]_{\mathrm{AllData}(W_2)}\big) \qquad (3)$$

We now use W_1 and W_2 and the monotonicity of $\mathcal{Q}^{P,S}_{c_{nf}}$ to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ (which proves that P is monotonic). W.l.o.g., let μ be an arbitrary solution for P in G_1, that is, $\mu \in [\![P]\!]_{G_1}$. Notice, such a μ must exist because we assume P is satisfiable (see before). To prove $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ it suffices to show $\mu \in [\![P]\!]_{G_2}$.

Due to Fact 5 it holds

$$\varrho[\mu] \in [\![P]\!]_{G_1'}$$

and with Facts 8 and 9 and Definition 12 we have

$$\varrho[\mu] \in \big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_1) \setminus [\![P^{*\prime}]\!]_{\mathrm{AllData}(W_1)}\big).$$

According to (3) we also have

$$\varrho[\mu] \in \big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_2) \setminus [\![P^{*\prime}]\!]_{\mathrm{AllData}(W_2)}\big).$$

We now use Definition 12 and Facts 9 and 8 again, to show

$$\varrho[\mu] \in [\![P]\!]_{G_2'}.$$

Finally, we use Fact 5 again and find

$$\varrho^{-1}\big[\varrho[\mu]\big] \in [\![P]\!]_{G_2}.$$

Since ϱ^{-1} is the inverse of bijective mapping ϱ, it holds $\varrho^{-1}\big[\varrho[\mu]\big] = \mu$ and, thus, we conclude $\mu \in [\![P]\!]_{G_2}$.

D Constant Reachability Criteria

This section discusses a particular class of reachability criteria which we call *constant reachability criteria*. These criteria always only accept a given, constant set of data links. As a consequence, each of these criteria ensures finiteness. In the following we formally introduce constant reachability criteria and prove that they ensure finiteness.

The (fixed) set of data links that a constant reachability criterion accepts may be specified differently. Accordingly, we distinguish four different types of constant reachability criteria. Formally, we define them as follows:

Definition 20. *Let $U \subset \mathcal{U}$ be a finite set URIs and let $T \subset \mathcal{T}$ be a finite set of RDF triples. The U-**constant reachability criterion** c^U is a reachability criterion that for each tuple $(t,u,P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ is defined as follows:*

$$c^U\Big(t,u,P\Big) = \begin{cases} \text{true} & \textit{if } u \in U, \\ \text{false} & \textit{else}. \end{cases}$$

*The T-**constant reachability criterion** c^T is a reachability criterion that for each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ is defined as follows:*

$$c^T\Big(t, u, P\Big) = \begin{cases} \text{true} & \textit{if } t \in T, \\ \text{false} & \textit{else}. \end{cases}$$

*The $(U \wedge T)$-**constant reachability criterion** $c^{U \wedge T}$ is a reachability criterion that for each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ is defined as follows:*

$$c^{U \wedge T}\Big(t, u, P\Big) = \begin{cases} \text{true} & \textit{if } u \in U \textit{ and } t \in T, \\ \text{false} & \textit{else}. \end{cases}$$

*The $(U \vee T)$-**constant reachability criterion** $c^{U \vee T}$ is a reachability criterion that for each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ is defined as follows:*

$$c^{U \wedge T}\Big(t, u, P\Big) = \begin{cases} \text{true} & \textit{if } u \in U \textit{ or } t \in T, \\ \text{false} & \textit{else}. \end{cases}$$

As can be seen from the definition, a U-constant reachability criterion uses a (finite) set U of URIs to specify the data links it accepts. Similarly, a T-constant reachability criterion uses a (finite) set T of RDF triples. $(U \wedge T)$-constant reachability criteria and $(U \vee T)$-constant reachability criteria combine U-constant reachability criteria and T-constant reachability criteria in a conjunctive and disjunctive manner, respectively. The reachability criterion c_{None} may be understood as a special case of U-constant reachability criteria; it uses a U which is empty. Similarly, c_{None} may be understood as the T-constant reachability criterion for which T is empty.

The following facts are trivial to verify:

Fact 10. *Let $U \subset \mathcal{U}$ and $U' \subset \mathcal{U}$ be finite sets of URIs such that $U' \subset U$. Similarly, let $T \subset \mathcal{T}$ and $T' \subset \mathcal{T}$ be finite sets of RDF triples such that $T' \subset T$. Furthermore, let c^U, $c^{U'}$, c^T, and $c^{T'}$ denote the U-constant reachability criterion, the U'-constant reachability criterion, the T-constant reachability criterion, and the T'-constant reachability criterion, respectively. Moreover, $c^{U \wedge T}$, $c^{U \vee T}$, $c^{U' \wedge T'}$, and $c^{U' \vee T'}$ denote the $(U \wedge T)$-constant reachability criterion, the $(U \vee T)$-constant reachability criterion, the $(U' \wedge T')$-constant reachability criterion, and the $(U' \vee T')$-constant reachability criterion, respectively. It holds:*

1. *$c^{U \vee T}$ is less restrictive than c^U and less restrictive than c^T.*
2. *c^U and c^T are less restrictive than $c^{U \wedge T}$, respectively.*
3. *c^U is less restrictive than $c^{U'}$.*
4. *c^T is less restrictive than $c^{T'}$.*
5. *$c^{U \wedge T}$ is less restrictive than $c^{U' \wedge T'}$.*
6. *$c^{U \vee T}$ is less restrictive than $c^{U' \vee T'}$.*

We now show that all constant reachability criteria ensure finiteness:

Proposition 13. *All U-constant, T-constant, $(U \wedge T)$-constant, and $(U \vee T)$-constant reachability criteria ensure finiteness.*

Proof of Proposition 13. To prove that a reachability criterion c ensures finiteness we have to show that for any Web of Linked Data W, any (finite) set $S \subset \mathcal{U}$ of seed URIs, and any SPARQL expression P, the (S, c, P)-reachable part of W is finite. W.l.o.g., let $S' \subset \mathcal{U}$ be an arbitrary (but finite) set of seed URIs, let P' be an arbitrary SPARQL expression. According to Definition 11 we know that the (S', c, P')-reachable part of any Web of Linked Data W is finite if the number of LD documents that are (c, P')-reachable from S' in W is finite. Due to the finiteness of S', it suffices to show that the set

$$X(c, P') = \left\{(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P} \,\middle|\, u \in \mathrm{uris}(t) \text{ and } P = P' \text{ and } c(t, u, P) = \mathrm{true}\right\}$$

is finite for any Web of Linked Data (cf. Definition 10). Notice, the given set presents an upper bound for all tuples $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ based on which LD documents may be reached by applying Definition 10 recursively. Hence, it is not necessarily the case that all these tuples are discovered (and used) during such a recursive application in a particular Web of Linked Data.

We now focus on U-constant, T-constant, $(U \wedge T)$-constant, and $(U \vee T)$-constant reachability criteria. W.l.o.g., let $U' \subset \mathcal{U}$ be an arbitrary, finite set of URIs and let $T' \subset \mathcal{T}$ be an arbitrary, finite set of RDF triples. Furthermore, let $c^{U'}$, $c^{T'}$, $c^{U' \wedge T'}$, and $c^{U' \vee T'}$ denote the U'-constant reachability criterion, the T'-constant reachability criterion, the $(U' \wedge T')$-constant reachability criterion, and the $(U' \vee T')$-constant reachability criterion, respectively.

For $c^{U' \vee T'}$ it holds

$$\left|X(c^{U' \vee T'}, P')\right| \leq |U'| + |T'|$$

and, thus, the set $X(c^{U' \vee T'}, P')$ is finite (recall, U' and T' are finite). Therefore, the $(S', c^{U' \vee T'}, P')$-reachable part of any Web of Linked Data is finite. As discussed before, this fact shows that $c^{U' \vee T'}$ ensures finiteness. However, we may also use this fact, together with Proposition 5, case 4, and Fact 10, cases 1 and 2, to show that $c^{U'}$, $c^{T'}$, and $c^{U' \wedge T'}$ ensure finiteness, respectively. □